LAIDD - Graph Neural Networks

Seongok Ryu, GALUX

강의 목차

1. (Lecture) Inductive biases, graph neural networks, attention and relational inference, Part. I
2. (Lecture) Inductive biases, graph neural networks, attention and relational inference, Part. II
3. (Tutorial) Installation of essential packages

   (Tutorial) Processing molecule data by using RDKit and TDC
4. (Lecture) Graph Convolutional Network (GCN), Graph Isomorphism Network (GIN)

   (Tutorial) GCN, GIN Implementation
5. (Lecture) Graph Attention Network (GAT)

   (Tutorial) GAT Implementation
6. (Lecture) Bayesian Deep Learning -- Reliability of molecular property prediction
7. (Tutorial) Bayesian Deep Learning Implementation,

   (Paper Review) Applications of GNN and Bayesian Learning in molecular supervised learning

강의 자료, 예시 코드

- github.com/seongokryu/LAIDD_GNN
- github.com/seongokryu/my-study-materials
- github.com/AITRICS/mol_reliable_gnn

Installation

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 dglteam 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

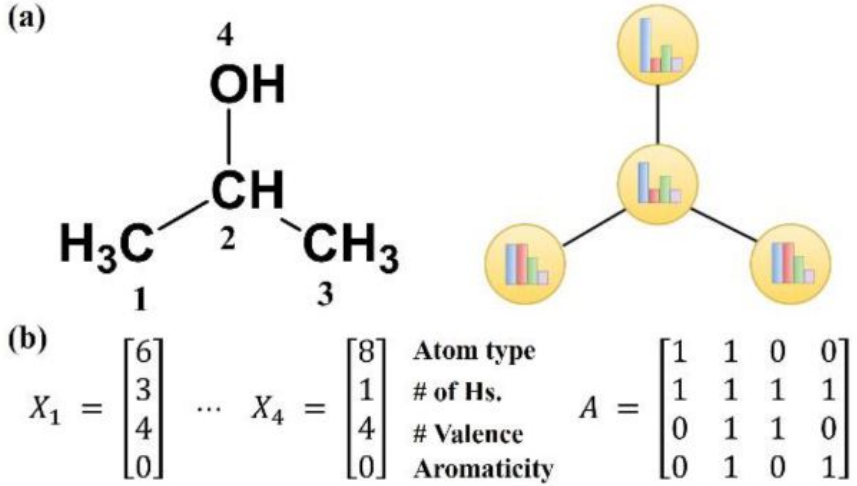
    → PIP install 명령어: pip install PyTDC

Graph Convolutional Network
Graph Isomorphism Network

Graph Convolutional Network

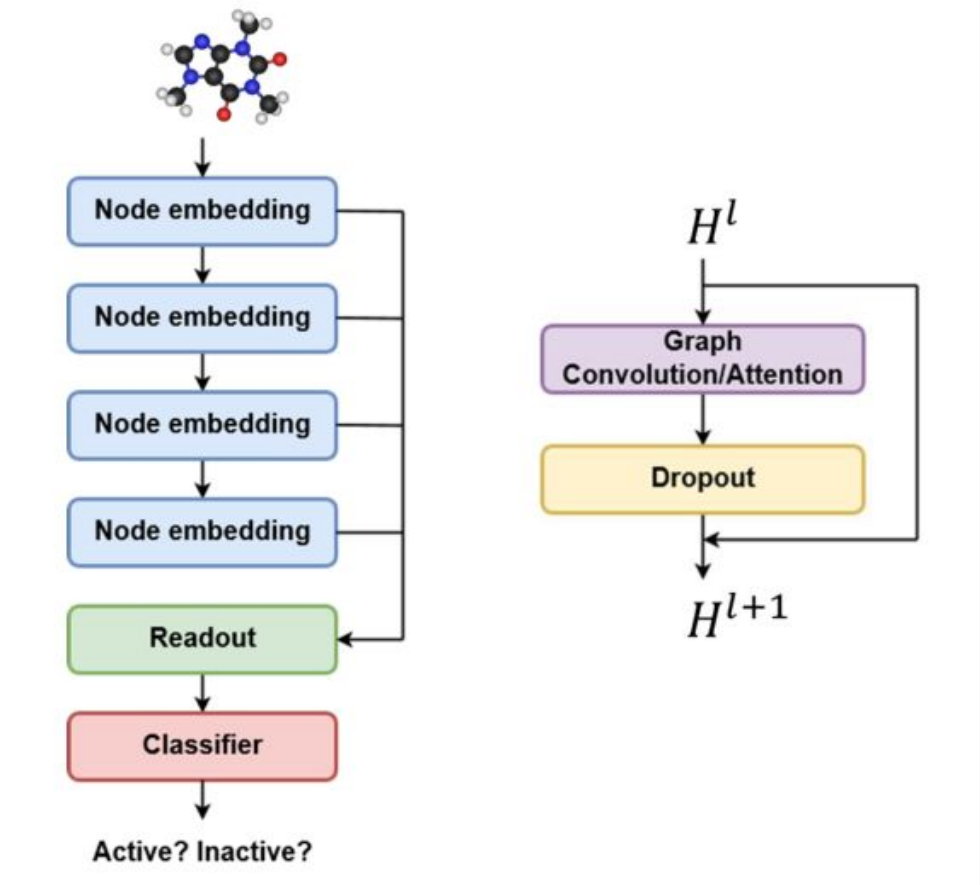
Preliminary) Molecular Graph

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



Input molecular graph of 2-propanol

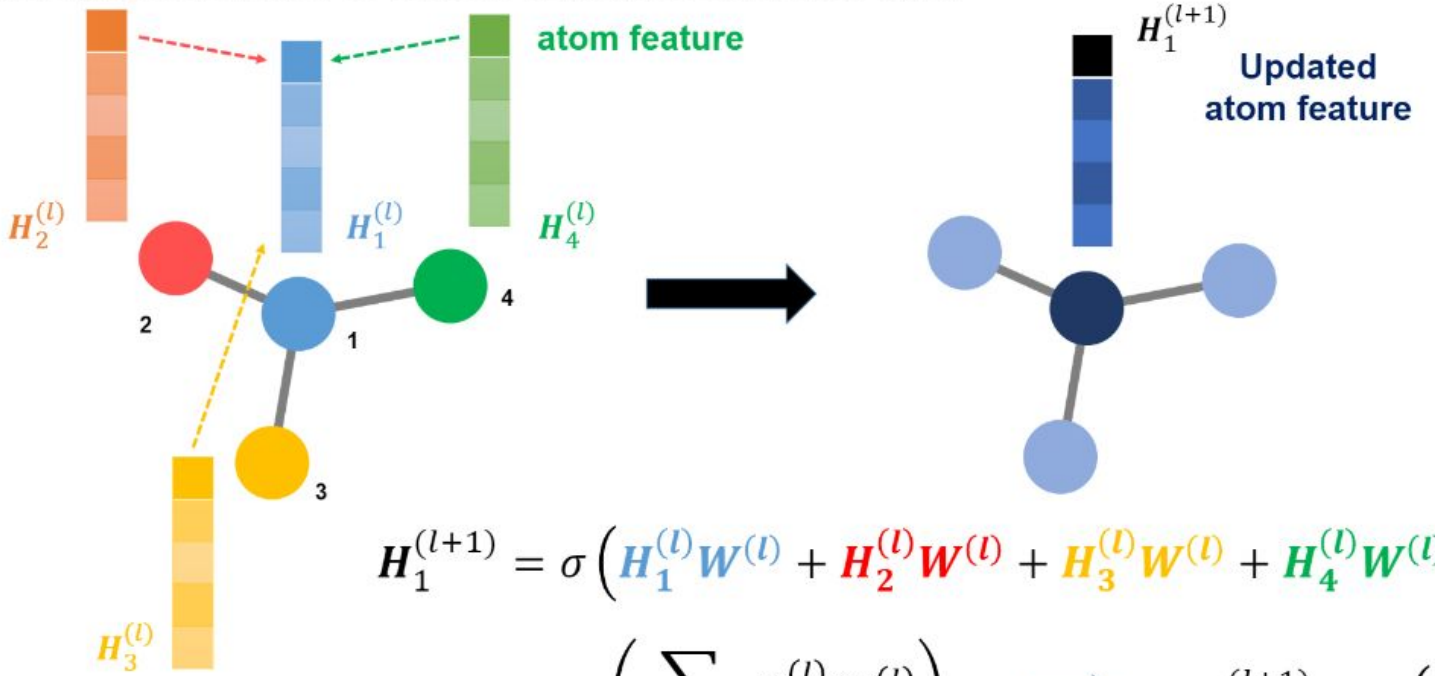
Graph Convolutional Network



Graph Convolutional Network

Preliminary) Graph convolutional networks

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



$$H_1^{(l+1)} = \sigma\left(H_1^{(l)}W^{(l)} + H_2^{(l)}W^{(l)} + H_3^{(l)}W^{(l)} + H_4^{(l)}W^{(l)}\right)$$

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

Graph Convolutional Network

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

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

- Average pooling

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

- Sum pooling

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

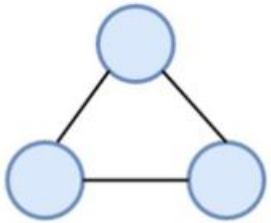
Graph Convolutional Network

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

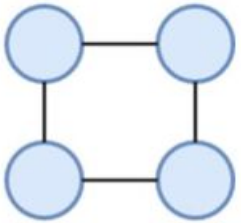
$$z^{(l)} = \text{Readout}\left(\left\{H_i^{(l)}\right\}\right)$$

- Average pooling cannot give correct summary statistics of node features

$$z_1 = \frac{1}{3}(h + h + h) = h \qquad z_2 = \frac{1}{4}(h + h + h + h) = h$$



$\mathcal{G}_1$



$\mathcal{G}_2$

- Sum pooling gives correct summary statistics of node features

$$z_1 = (h + h + h) = 3h \qquad z_2 = (h + h + h + h) = 4h$$

Graph Convolutional Network

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

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

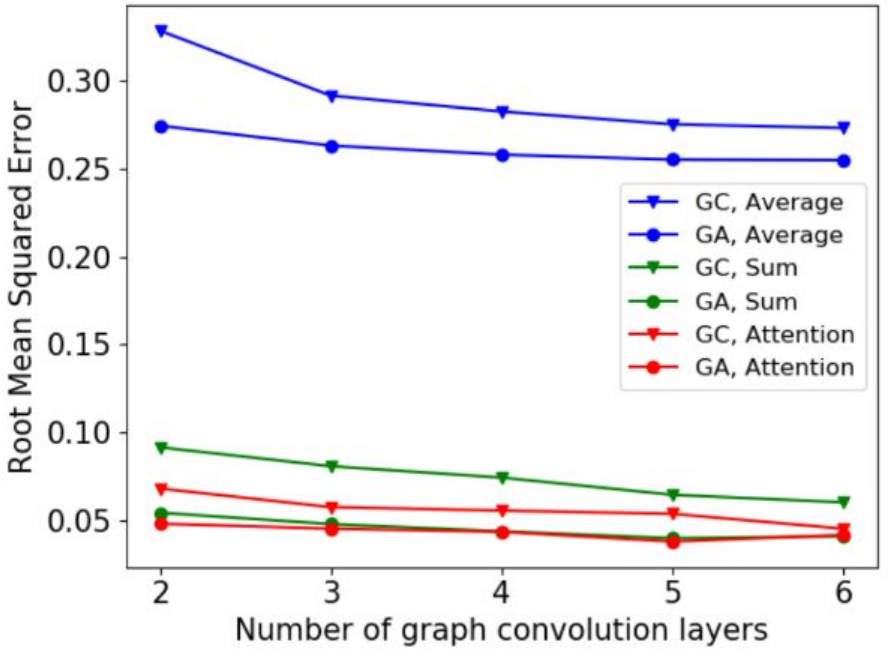
- Concatenation can summarize hierarchical sub-graph structures

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

Graph Convolutional Network

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

In preparation

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	**0.043**	0.055	**0.52**	0.60	**0.053**	0.054

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

Graph Convolutional Network

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 (↑)	AUROC (↑)	F1-score (↑)	ECE (↓)	OCE (↓)
BACE	GCN+Sum	0.809	0.878	0.780	0.102	0.086
	GCN+Attn	**0.822**	**0.897**	**0.802**	**0.091**	**0.065**
	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	**0.066**	**0.082**
	GCN+Attn	**0.892**	**0.919**	**0.931**	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	**0.008**	0.008
	GCN+Attn	**0.971**	**0.816**	**0.438**	0.010	**0.007**
	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)

Key Paper

- 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

Graph Isomorphism Network

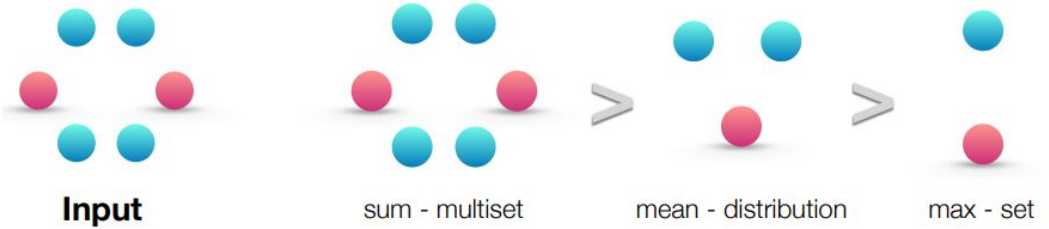
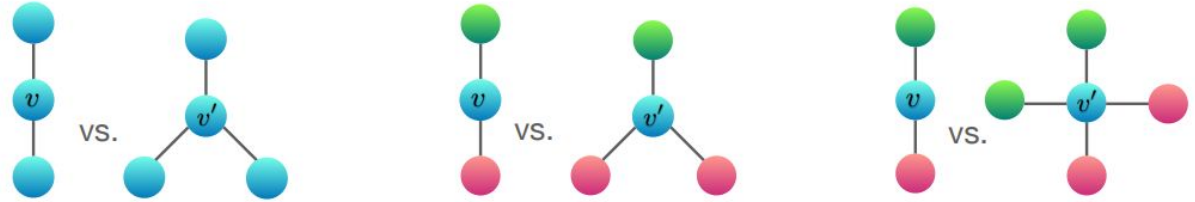


Figure 2: **Ranking by expressive power for sum, mean and max aggregators over a multiset**. Left panel shows the input multiset, *i.e.*, the network neighborhood to be aggregated. The next three panels illustrate the aspects of the multiset a given aggregator is able to capture: sum captures the full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



(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

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.

Graph Isomorphism Network

Corollary 6. *Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that for infinitely many choices of ϵ, 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 $g(c, X) = \varphi\left((1+\epsilon) \cdot f(c) + \sum_{x \in X} f(x)\right)$ for some function φ.*

Universal function approximator : MLP

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

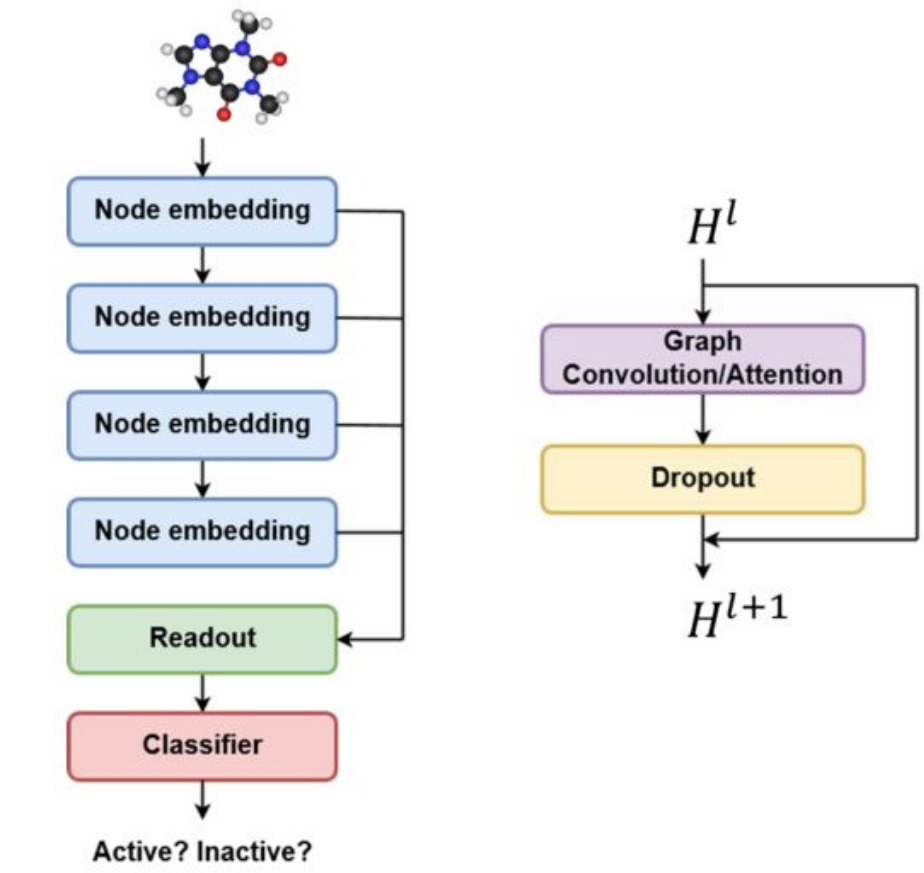
- GCN: Linear layer and non-linear activation (e.g. ReLU)

  → Less expressive power than GIN

Let’s practice implementation

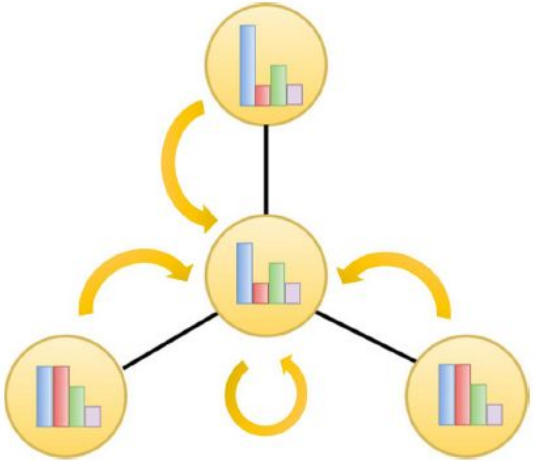
Graph Attention Network

GNN architecture



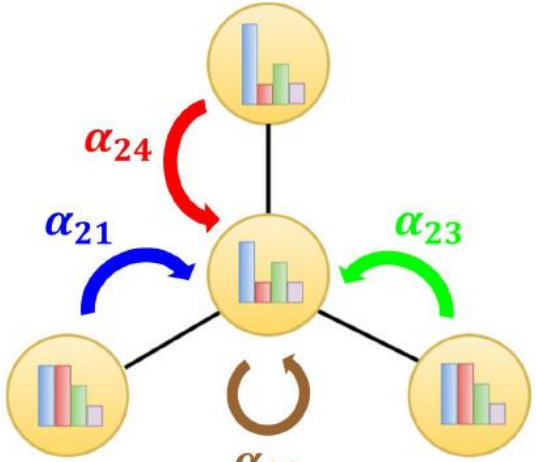
GCN vs GAT

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



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

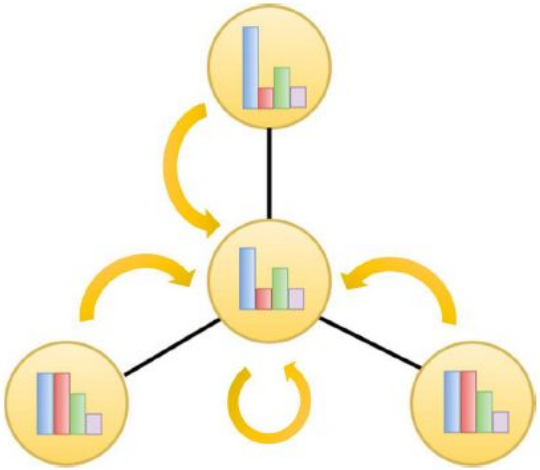
Attention mechanism enables GCN to update nodes **with different importance**.



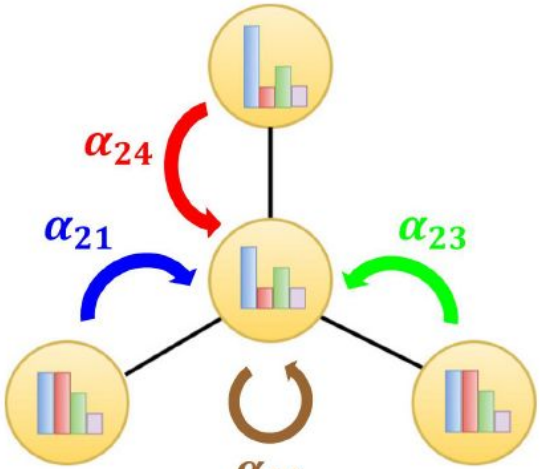
$$H^{(l+1)} = \sigma\left(\sum_{j \in N(i)} \alpha_{ij} H_j^{(l)} W^{(l)}\right)$$

GCN vs GAT

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



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

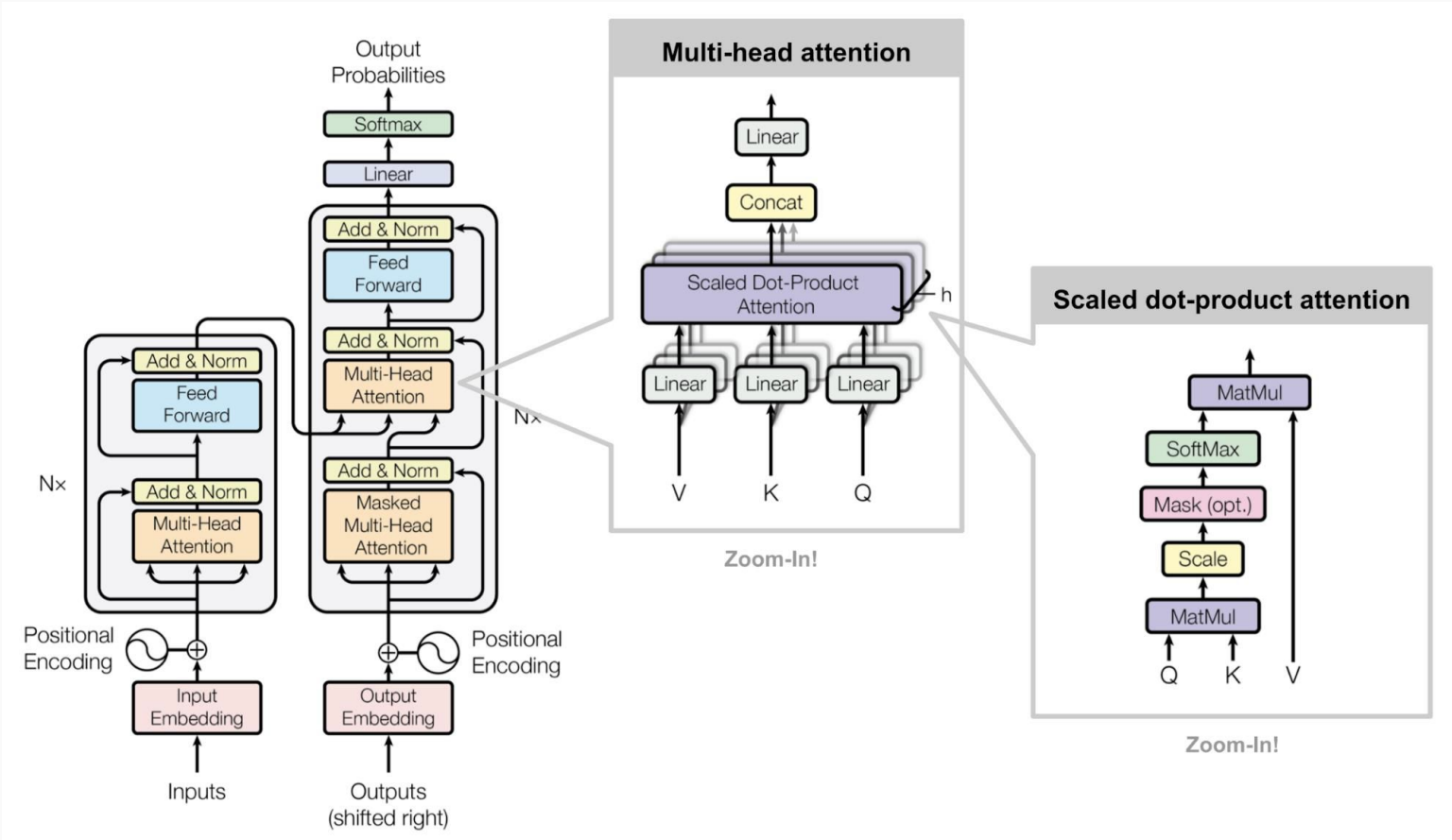


$$H^{(l+1)} = \sigma\left(\sum_{j \in N(i)} \alpha_{ij} H_j^{(l)} W^{(l)}\right)$$

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

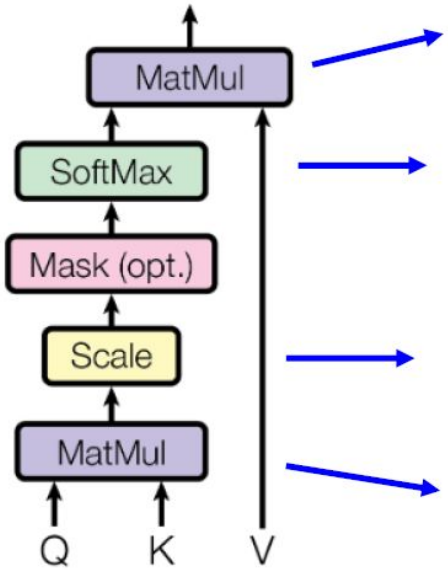
Self Attention - Transformer



Self Attention - Transformer

Self-attention

Scaled Dot-Product Attention



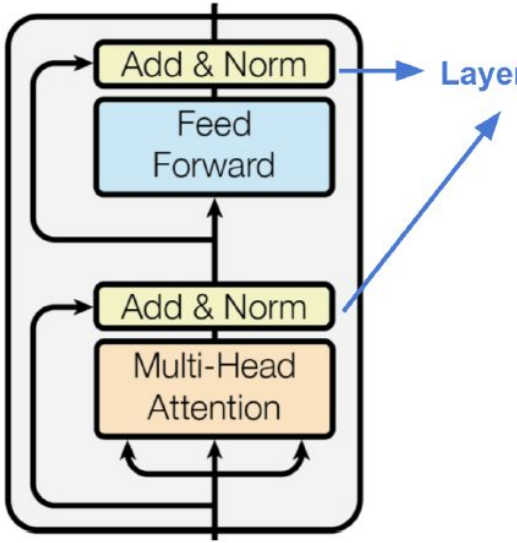
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \in \mathbb{R}^{d_{model} \times d_v}$$

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

$\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}$: 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



N×
= 6

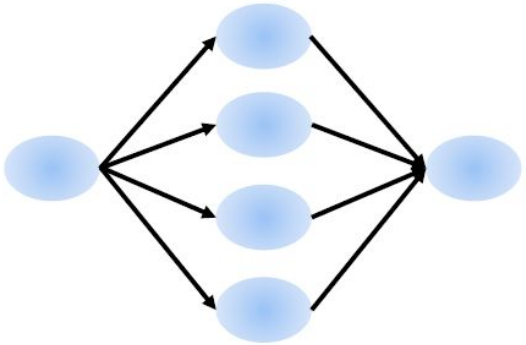
LayerNorm(x + SubLayer(x))

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

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

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

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



GAT -- original paper, Petar Velickovic, et. al., ICLR 2018

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

Guillem Cucurull*
Centre de Visió per Computador, UAB
gcucurull@gmail.com

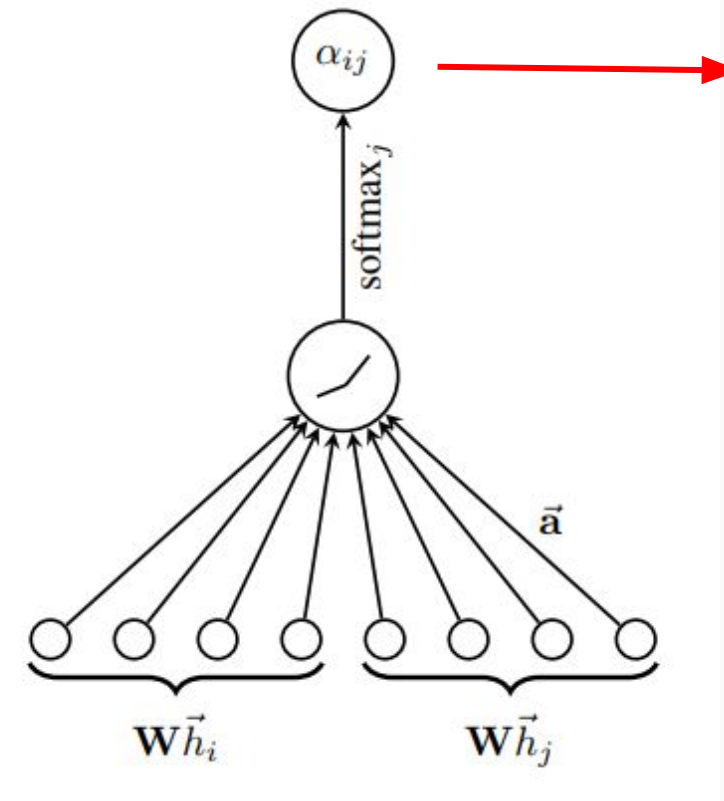
Arantxa Casanova*
Centre de Visió per Computador, UAB
ar.casanova.8@gmail.com

Adriana Romero
Montréal Institute for Learning Algorithms
adriana.romero.soriano@umontreal.ca

Pietro Liò
Department of Computer Science and Technology
University of Cambridge
pietro.lio@cst.cam.ac.uk

Yoshua Bengio
Montréal Institute for Learning Algorithms
yoshua.umontreal@gmail.com

GAT -- original paper, Petar Velickovic, et. al., ICLR 2018

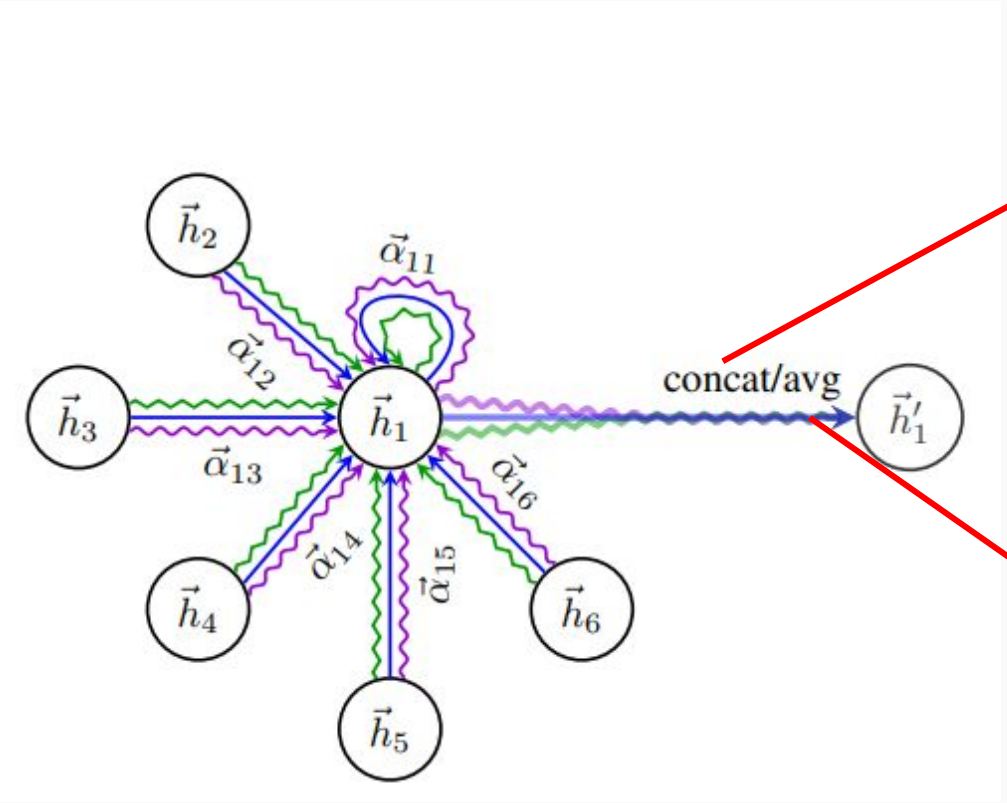


$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}.$$

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

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

GAT -- original paper, Petar Velickovic, et. al., ICLR 2018



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

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

GAT -- original paper, Petar Velickovic, et. al., ICLR 2018

- 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

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]

PyTorch geometric

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

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

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**

Let’s practice implementation

Bayesian Learning

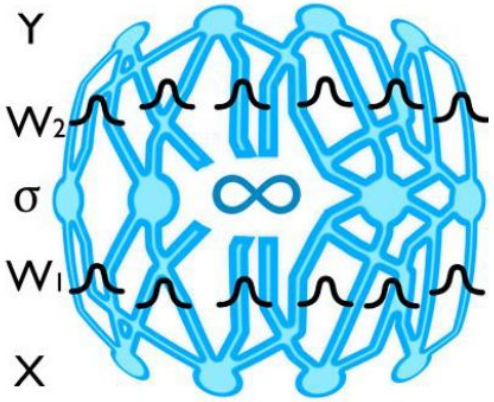
Statistical Inference and Bayesian Inference

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

Statistical Inference and Bayesian Inference

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

likelihood prior

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

evidence (or marginal likelihood)

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

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

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

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

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: $\hat{\boldsymbol{\omega}} = \underset{\boldsymbol{\omega} \in \Omega}{\operatorname{argmax}} \mathcal{L}(\boldsymbol{\omega})$
- Maximum-a-posteriori (MAP) estimation: $\hat{\boldsymbol{\omega}} = \underset{\boldsymbol{\omega} \in \Omega}{\operatorname{argmax}} p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})\, p(\boldsymbol{\omega})$
- Maximum-likelihood estimation (MLE) : $\hat{\boldsymbol{\omega}} = \underset{\boldsymbol{\omega} \in \Omega}{\operatorname{argmax}} p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})$

Assuming prior has a uniform distribution

- Frequentist inference: $\mathbf{y}^* = \mathrm{f}^{\hat{\boldsymbol{\omega}}}(\mathbf{x}^*)$

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

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

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

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.

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

Then, our inference will be a single deterministic value.

Inference from Bayesian model estimation

Summation over all possible model posteriors

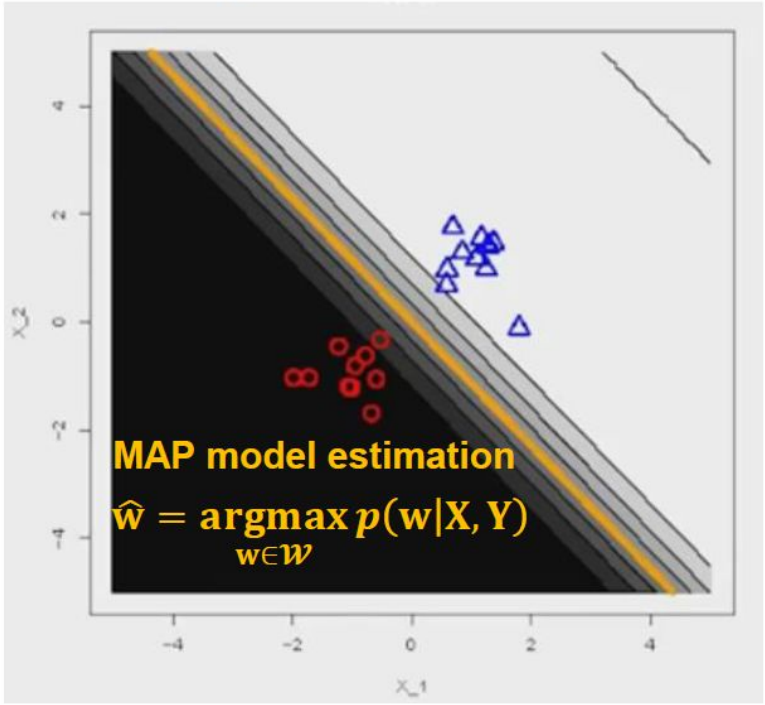
$$p(y^*|x^*, X, Y) = \int_{w \in \mathcal{W}} p(y^*|x^*, w) p(w|X, Y) dw$$

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

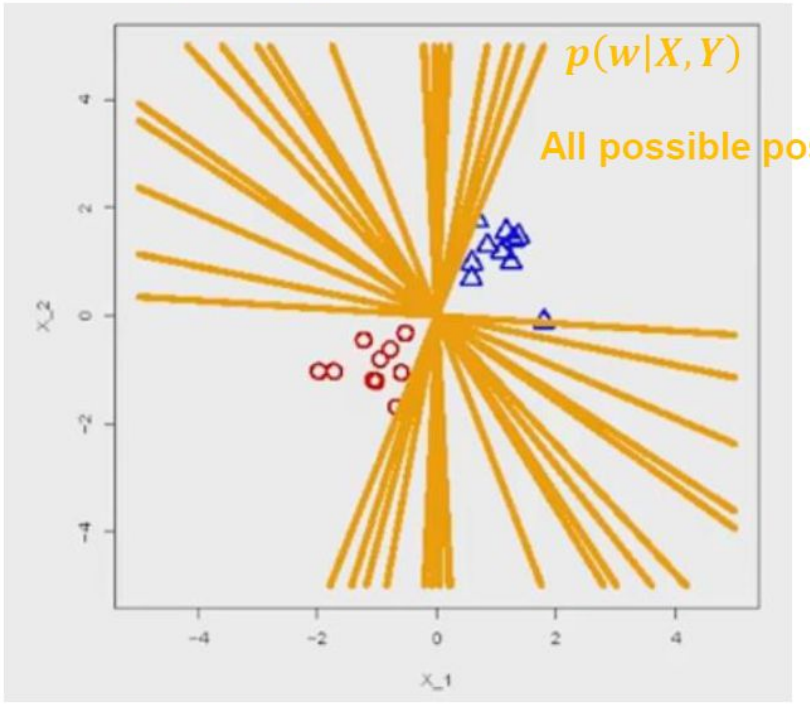
→ The uncertainty on model parameter give the uncertainty on our inference

Statistical Inference and Bayesian Inference

MAP predictive distribution



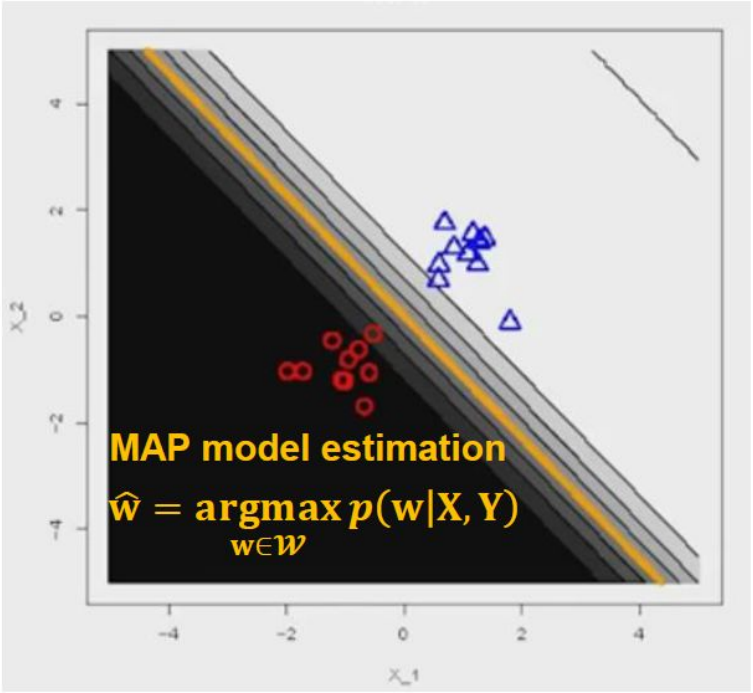
Bayesian predictive distribution



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

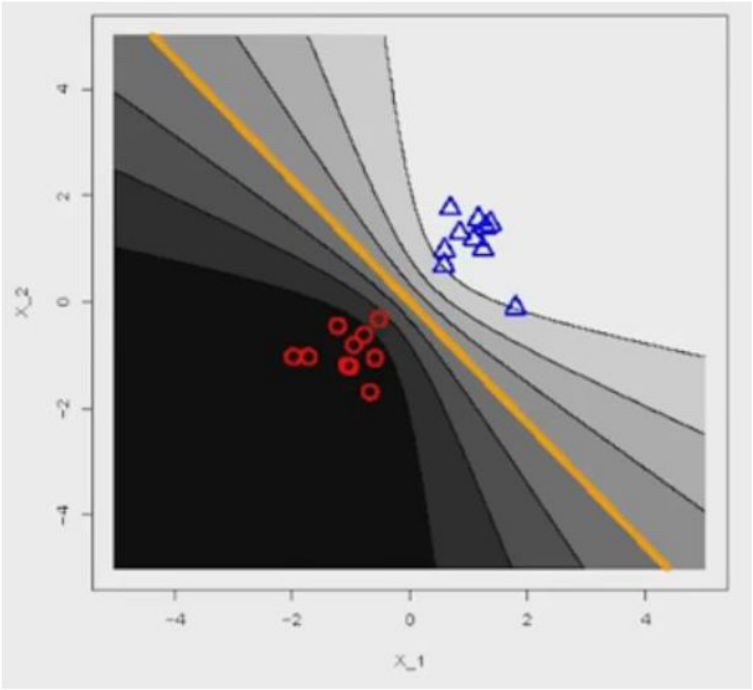
Statistical Inference and Bayesian Inference

MAP predictive distribution



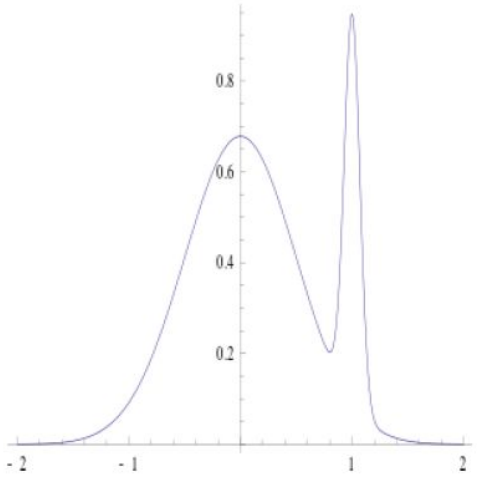
$$p(y^*|x^*) = f^{\hat{w}}(x^*)$$

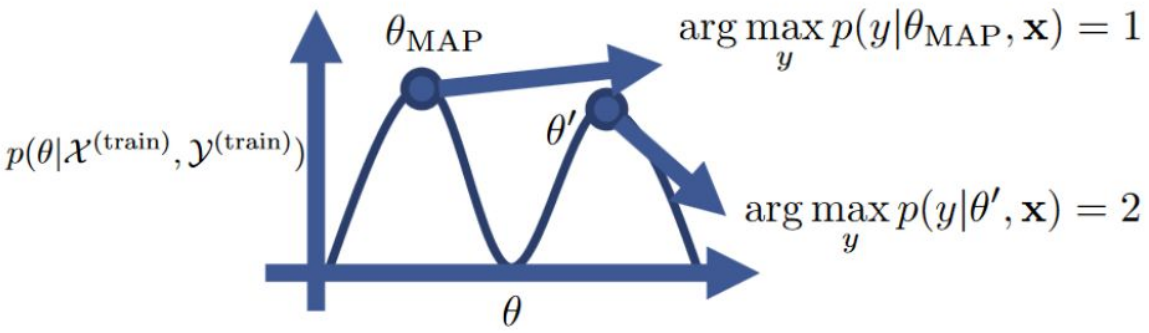
Bayesian predictive distribution



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

Limitations of Frequentist's model estimation:





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

Limitations of Bayesian's model estimation:

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

likelihood prior

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

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

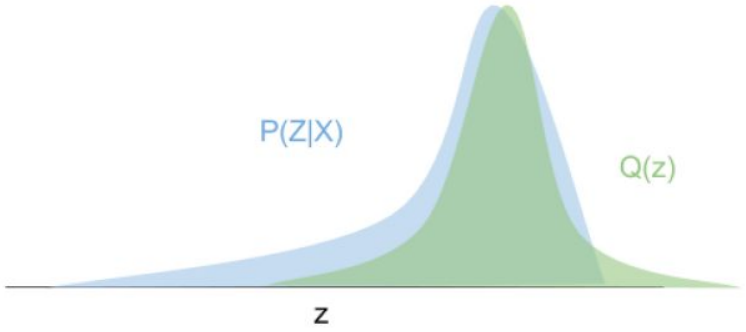
  : good approximation is needed, but it often hurts the quality of approximated posterior.

Variational Inference

Reparameterization trick using variational parameter θ

$$p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y}) \approx q_\theta\ (\boldsymbol{\omega})$$

Kullback-Leibeler divergence
: metric to make two distributions similar



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

Still intractable because the posterior exists in the KL term.

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

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

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

Variational Inference

Therefore, our minimization objective is

$$\mathrm{KL}\big(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})\big) = -\int_\Omega q_\theta(\boldsymbol{\omega})\log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))d\boldsymbol{\omega} + \mathrm{KL}\big(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega})\big)$$

$$= -\sum_{i=1}^{N}\int_\Omega q_\theta(\boldsymbol{\omega})\log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))d\boldsymbol{\omega} + \mathrm{KL}\big(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega})\big)$$

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

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

$$\rightarrow \frac{1}{M}\sum_{i\in S}^{M}\int_\Omega q_\theta(\boldsymbol{\omega})\log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))d\boldsymbol{\omega} + \frac{1}{N}\mathrm{KL}\big(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega})\big)$$

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.

$$\mathcal{L}_{VI}(\theta) = -\frac{1}{M}\sum_{i\in S}\int_{\Omega} \log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))\, q_\theta(\boldsymbol{\omega})d\boldsymbol{\omega} + \frac{1}{N}\mathrm{KL}(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega}))$$

$$= -\frac{1}{M}\sum_{i\in S}\int_{\Omega} \log p\left(\mathbf{y}_i\middle|\mathbf{f}^{g(\theta,\boldsymbol{\epsilon})}(\mathbf{x}_i)\right) p(\boldsymbol{\epsilon})d\boldsymbol{\epsilon} + \frac{1}{N}KL(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega}))$$

$$\hat{\mathcal{L}}_{MC}(\theta) = -\frac{1}{M}\sum_{i\in S}\log p\left(\mathbf{y}_i\middle|\mathbf{f}^{g(\theta,\boldsymbol{\epsilon})}(\mathbf{x}_i)\right) + \frac{1}{N}\mathrm{KL}(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega}))$$

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

$$\tilde{q}_\theta(\mathbf{y}^*|\mathbf{x}^*) \coloneqq \frac{1}{T}\sum_{t=1}^{T} p(\mathbf{y}^*|\mathbf{x}^*, \hat{\boldsymbol{\omega}}_t) \xrightarrow[T\to\infty]{} \int p(\hat{\mathbf{y}}^*|\hat{\mathbf{x}}^*, \boldsymbol{\omega})q_\theta(\boldsymbol{\omega})d\boldsymbol{\omega}$$

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

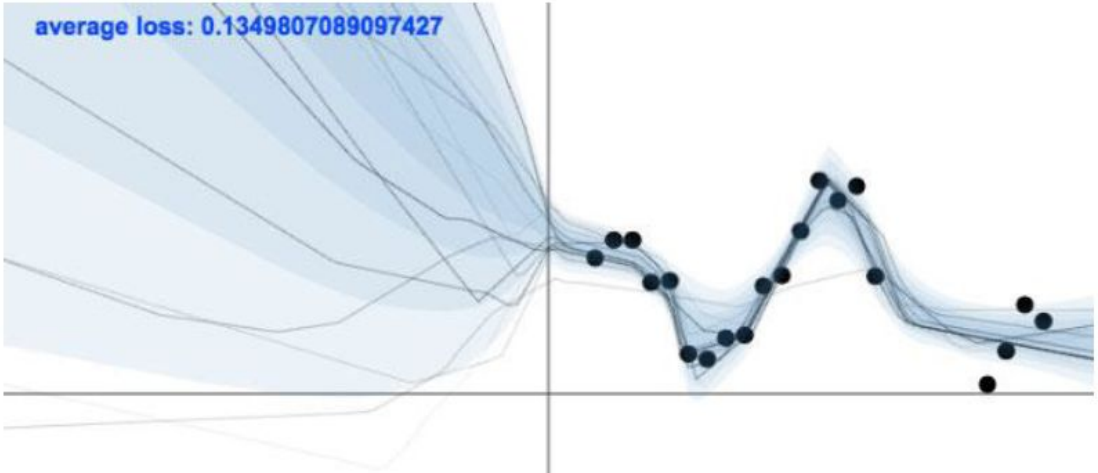
$$= p(\hat{\mathbf{y}}^*|\hat{\mathbf{x}}^*, \mathbf{X}, \mathbf{Y})$$

Bayesian Uncertainty

Bayesian inference and uncertainty

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

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



"Knowing what we don't know" is as important as *"knowing itself"*.

Bayesian Uncertainty

Homoskedastic regression assumes constant noise σ for every input point $\mathbf{x}$.

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

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

Heteroskedastic regression, on the other hand, assumes that observation noise can vary with input $\mathbf{x}$.

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

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

To summarize, the predictive uncertainty can be approximated using:

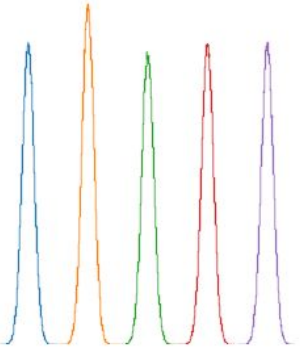
$$Var(y) \approx \frac{1}{T}\sum_{t=1}^{T}\hat{y}_t^2 - \left(\frac{1}{T}\sum_{t=1}^{T}\hat{y}_t\right)^2 + \frac{1}{T}\sum_{t=1}^{T}\hat{\sigma}_t^2$$

Epistemic uncertainty **Aleatoric uncertainty**

Bayesian Uncertainty

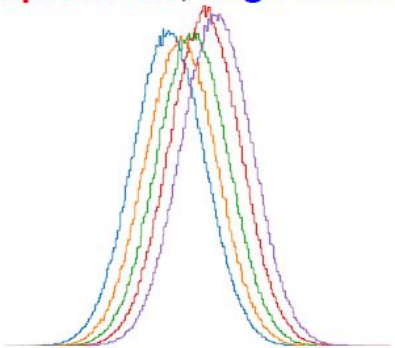
Low epistemic, Low aleatoric

High epistemic, Low aleatoric

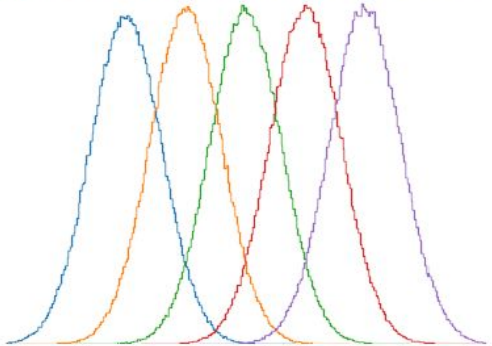


$$Var(y) \approx \frac{1}{T}\sum_{t=1}^{T}\hat{y}_t^2 - \left(\frac{1}{T}\sum_{t=1}^{T}\hat{y}_t\right)^2 + \frac{1}{T}\sum_{t=1}^{T}\hat{\sigma}_t^2$$

Low epistemic, High aleatoric



High epistemic, High aleatoric



Bayesian Inference in practice: MC-Dropout

Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning **ICML 2015**

Yarin Gal YG279@CAM.AC.UK
Zoubin Ghahramani ZG201@CAM.AC.UK
University of Cambridge

```python
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 phase 에서 Dropout 을 키고 T번 output을 sampling (MC-sampling)
- 오른쪽의 식과 같이 regression/classification 에 대해서 predictive mean/uncertainty를 계산

Bayesian Inference in practice: Stochastic Weight Averaging

Averaging Weights Leads to Wider Optima and Better Generalization

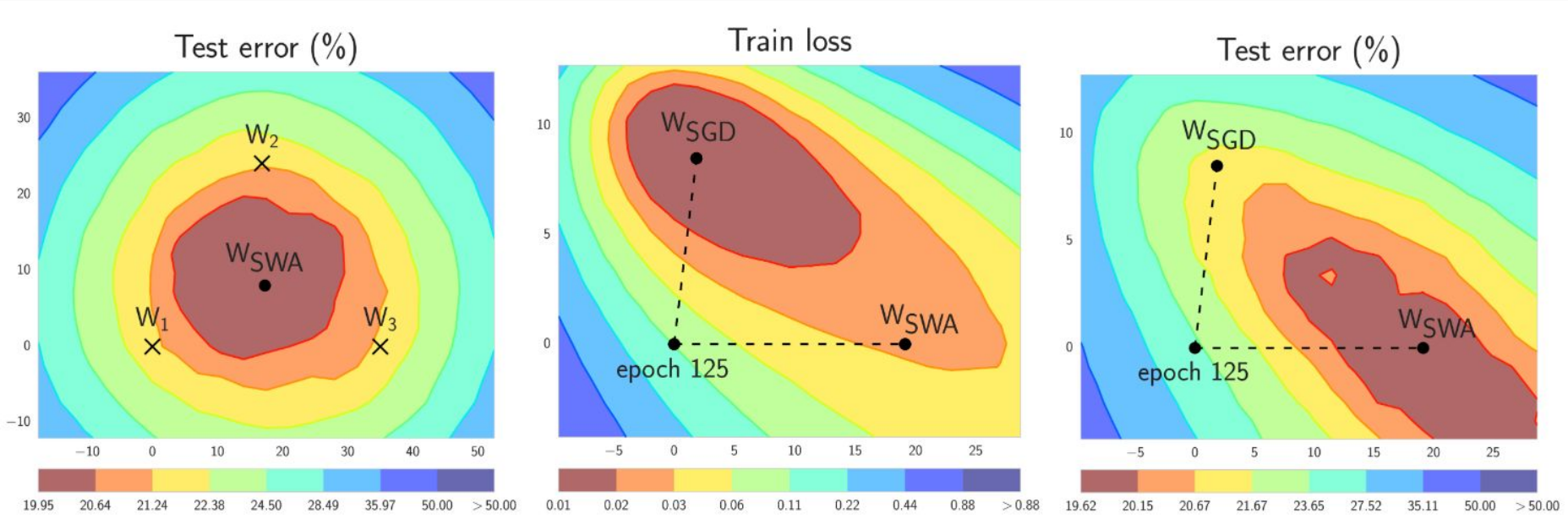
Pavel Izmailov[*1] **Dmitrii Podoprikhin**[*2,3] **Timur Garipov**[*4,5] **Dmitry Vetrov**[2,3] **Andrew Gordon Wilson**[1]
[1]Cornell University, [2]Higher School of Economics, [3]Samsung-HSE Laboratory,
[4]Samsung AI Center in Moscow, [5]Lomonosov Moscow State University

April 29, 2019

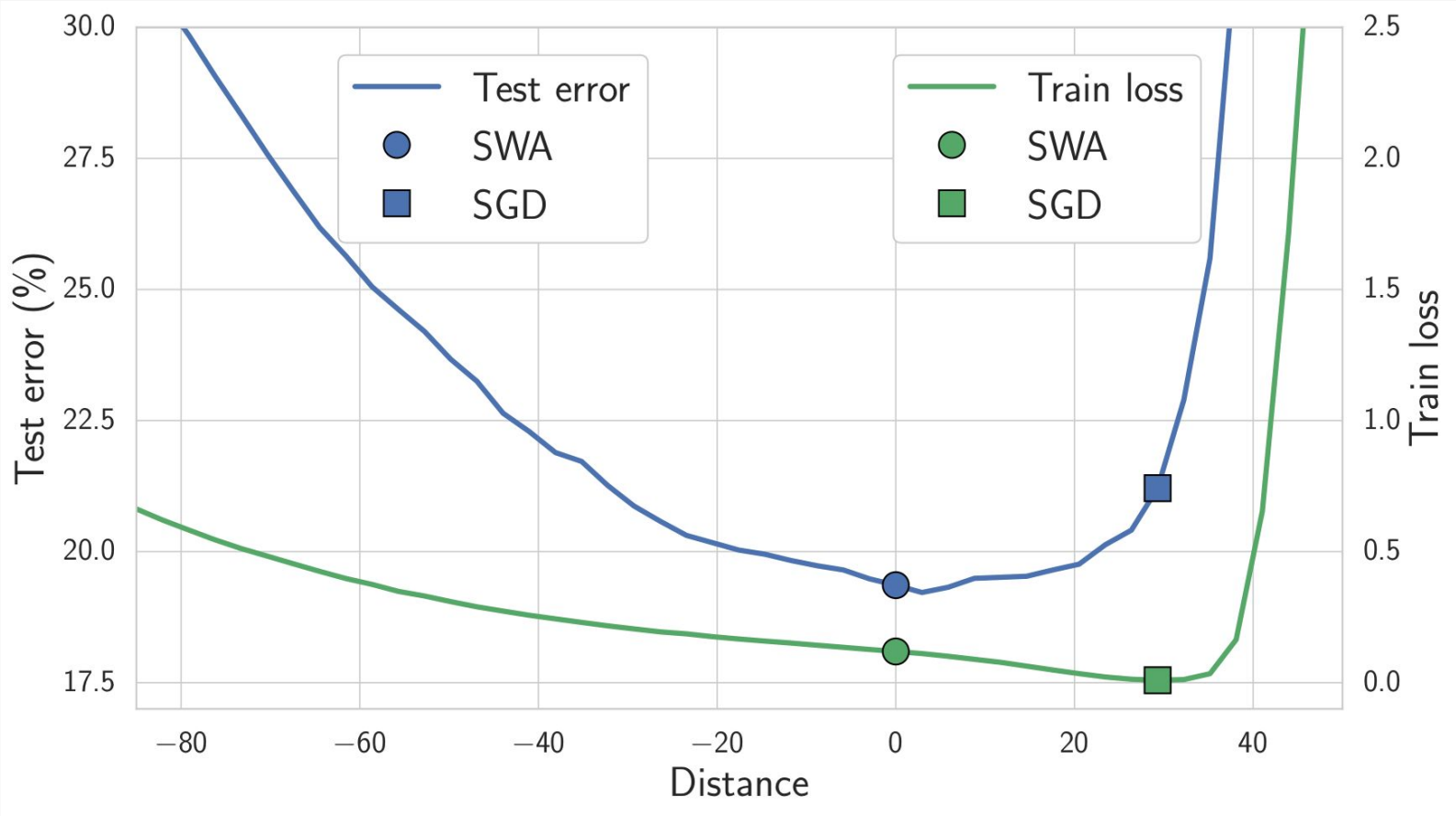
Stochastic Weight Averaging in PyTorch

Official PyToch Blog: https://pytorch.org/blog/stochastic-weight-averaging-in-pytorch/

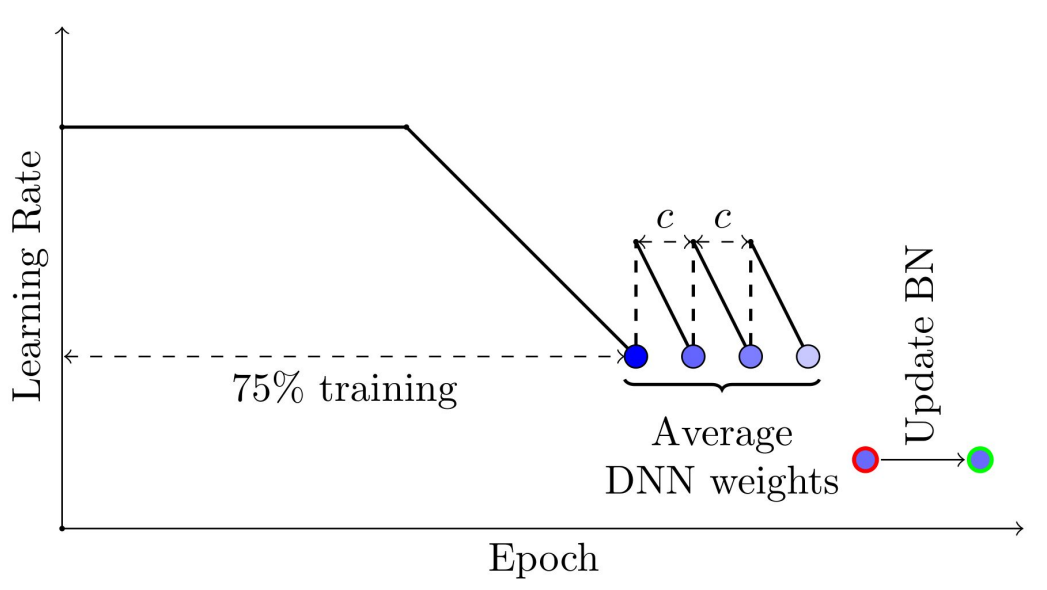
Bayesian Inference in practice: Stochastic Weight Averaging



Bayesian Inference in practice: Stochastic Weight Averaging



Bayesian Inference in practice: Stochastic Weight Averaging



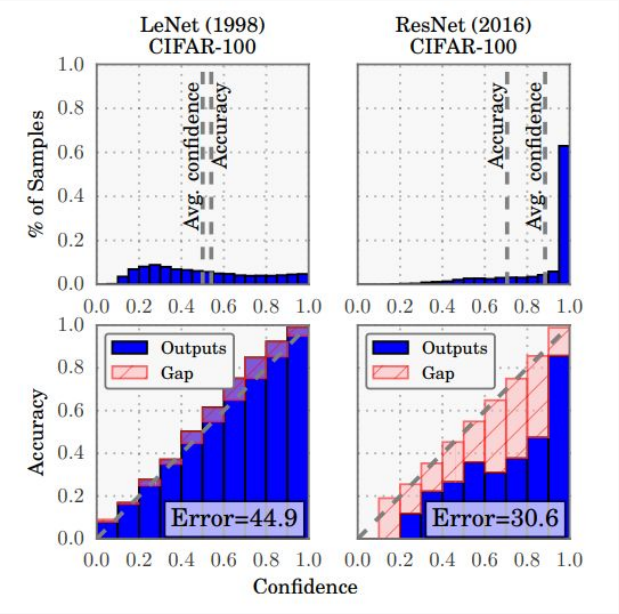

```
opt = torchcontrib.optim.SWA(base_opt)
for i in range(100):
    opt.zero_grad()
    loss_fn(model(input), target).backward()
    opt.step()
    if i > 10 and i % 5 == 0:
        opt.update_swa()
opt.swap_swa_sgd()
```

- 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 [*1] Geoff Pleiss [*1] Yu Sun [*1] Kilian Q. Weinberger [1]

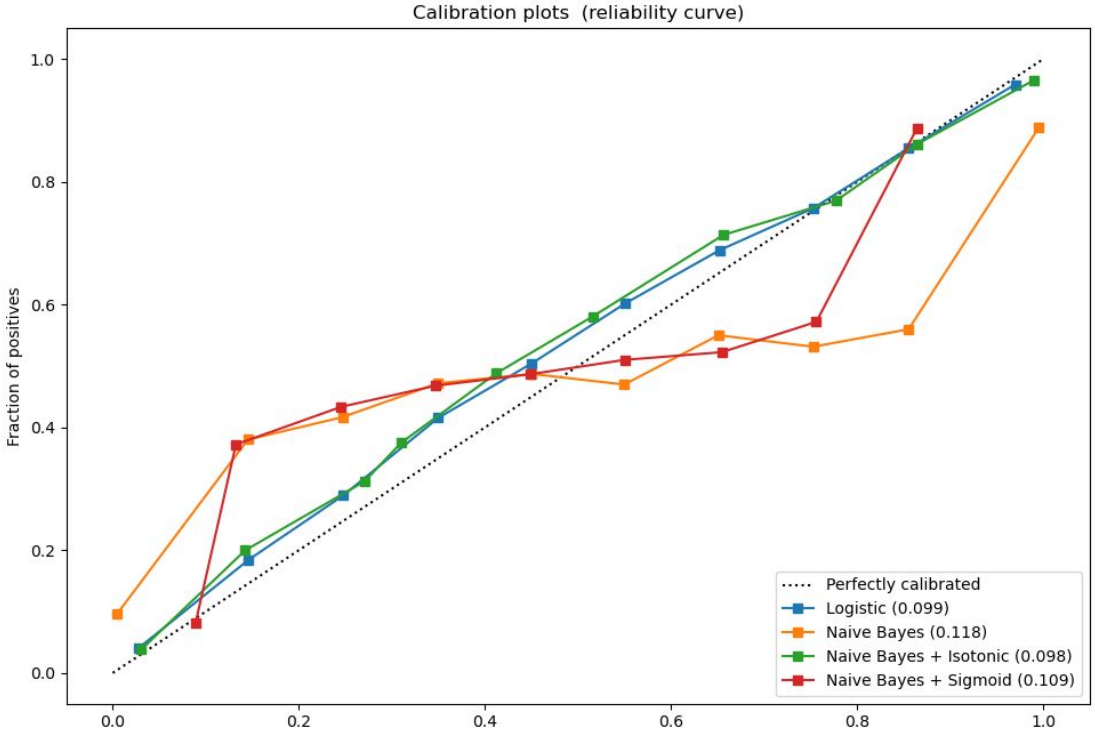


- 초기 CNN 구조인 LeNet (1998) 에 비해 ResNet (2016) 은 Accuracy가 많이 개선됨. (Error: 44.9% vs 30.6% for CIFAR-100)
- 하지만 모델 parameter 갯수가 크게 증가하였고,
- 특정 class에 속할 확률을 over-estimation하는 경향이 있음

→ Over-confident prediction problem

- "어떻게 모델의 over-confidence 여부, 즉 reliability (신뢰도) 를 평가할 수 있을까?"

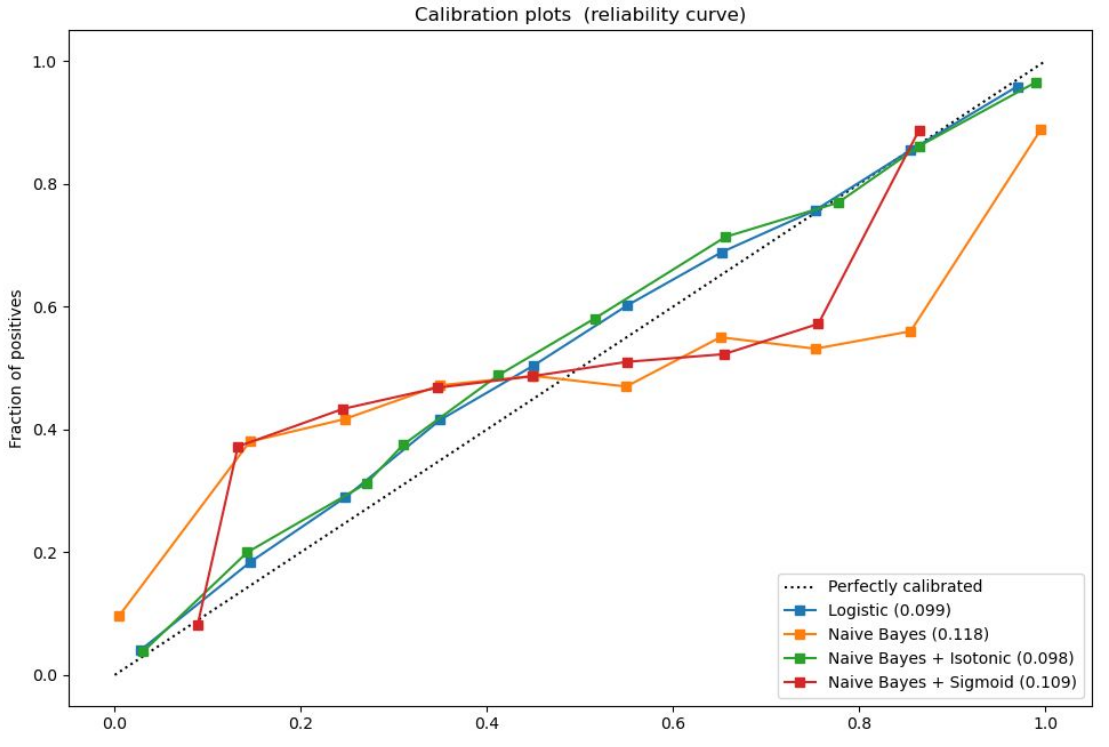
How to evaluate reliability of models? -- Expected Calibration Error



- 점선: perfect calibration line
- Output 값이 0.7이면 class=1 일 확률이 70%, 0.9이면 class=1일 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

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



Perfect calibration case)

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

Perfect calibration과 실제 model results의 차이 → calibration error

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

실제 histogram 방식으로 구하는 경우

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

https://scikit-learn.org/stable/auto_examples/calibration/plot_calibration_curve.html

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

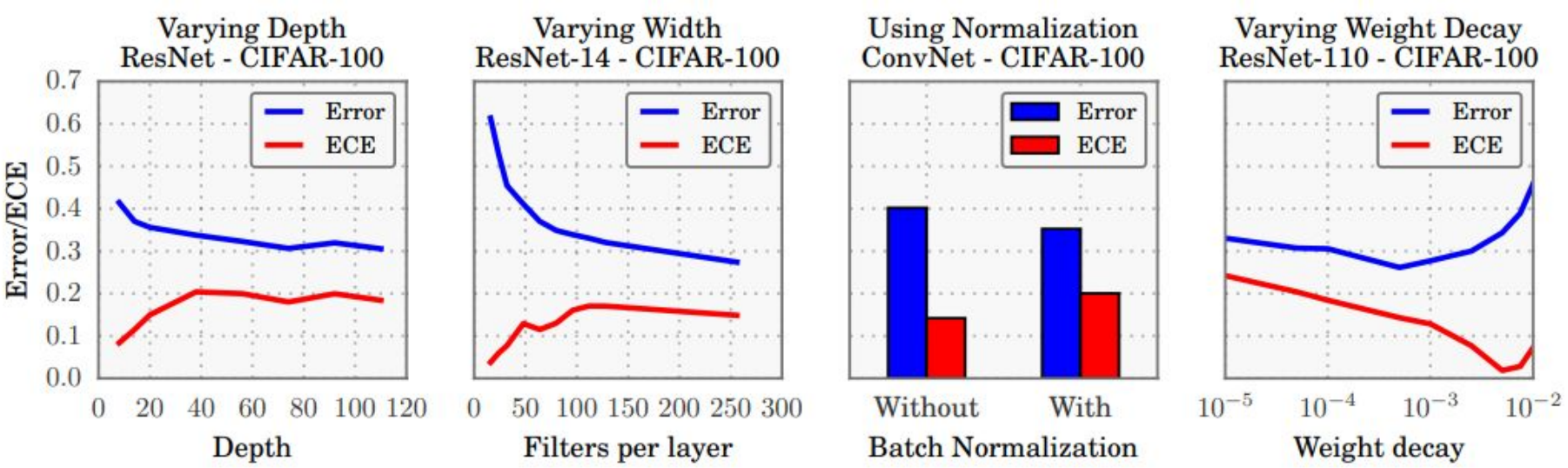


Figure 2. The effect of network depth (far left), width (middle left), Batch Normalization (middle right), and weight decay (far right) on miscalibration, as measured by ECE (lower is better).

- 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

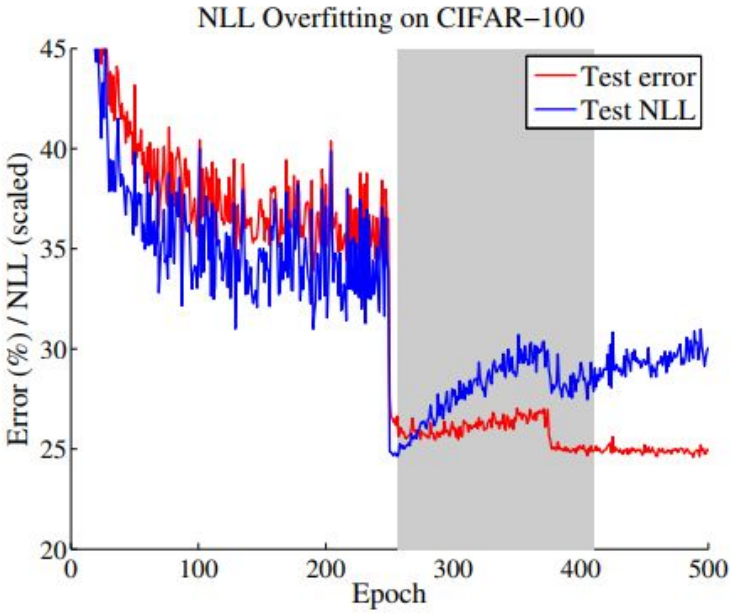


Figure 3. Test error and NLL of a 110-layer ResNet with stochastic depth on CIFAR-100 during training. NLL is scaled by a constant to fit in the figure. Learning rate drops by 10x at epochs 250 and 375. The shaded area marks between epochs at which the best validation *loss* and best validation *error*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

- 모델을 계속 트레이닝하다보면 (training epoch이 지나다보면)
  - Test error는 어느순간부터 더 떨어지지 않음.
  - 하지만 negative log-likelihood (NLL) 은 증가함.
  - 이런 결과는 reliability를 떨어트리는 결과로 이어짐

    → SWA는 이런 NLL over-fitting을 방지함.

Applications in molecular property predictions

Chemical Science

ROYAL SOCIETY OF CHEMISTRY

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,[a] Yongchan Kwon [b] and Woo Youn Kim *[ac]

Applications in molecular property predictions

JCIM JOURNAL OF CHEMICAL INFORMATION AND MODELING

pubs.acs.org/jcim Article

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