Diffusion Model Applications and Bayesian Neural Networks

CSE 849 Deep Learning
Spring 2025

Zijun Cui

Slides are revised based on Deep Learning CMU by Hao Chen and Probabilistic Deep Learning RPI by Qiang Ji

Content

- Diffusion Model Basics
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Variants of Diffusion Models
  - Denoising Diffusion Implicit Model (DDIM)
  - Latent Diffusion Models
  - Conditional Diffusion Models
- Applications of Diffusion Models

Case study: Imagen

Imagen: text-to-image diffusion models

By Google (imagen.research.google)

Input: text; Output: 1kx1k images

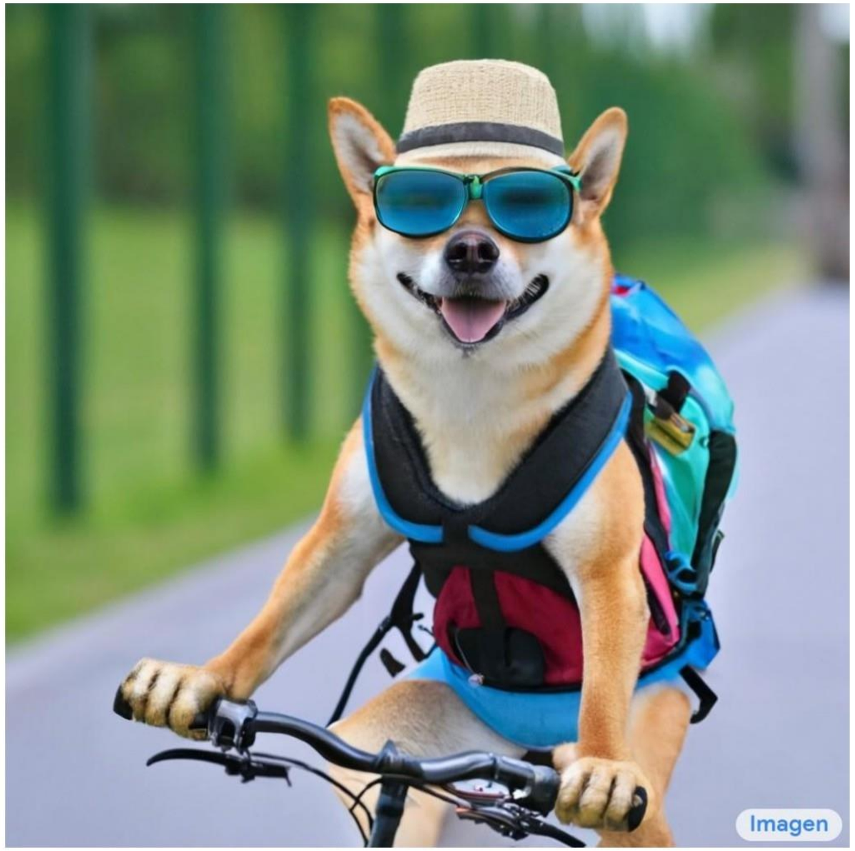
- An unprecedented degree of photorealism
  - SOTA automatic scores & human ratings
- A deep level of language understanding
- Extremely simple
  - no latent space, no quantization

A brain riding a rocketship heading towards the moon.

Saharia et al., “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, arXiv 2022.

Imagen

By Google (imagen.research.google)



A photo of a Shiba Inu dog with a backpack riding a bike. It is wearing sunglasses and a beach hat.

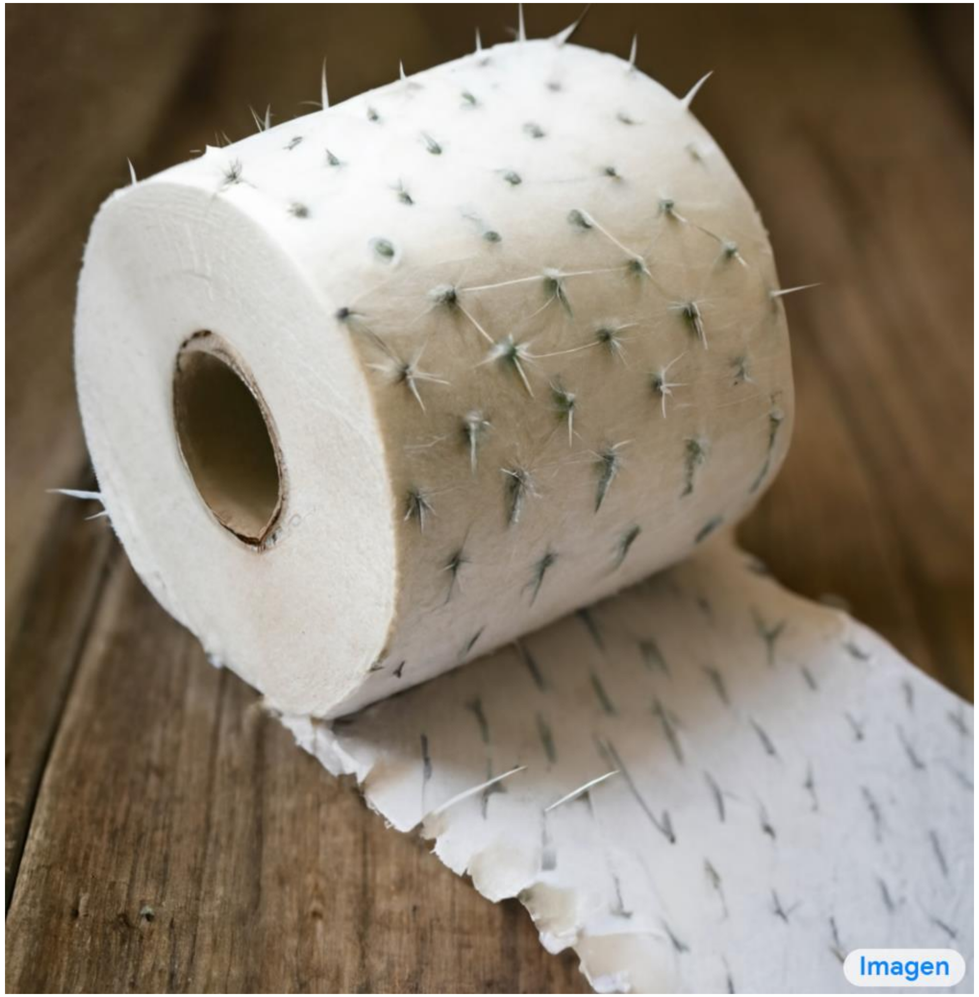
Saharia et al., “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, arXiv 2022.

Imagen

By Google (imagen.research.google)

A dragon fruit wearing karate belt in the snow.

Saharia et al., "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", arXiv 2022.



“toilet paper with real cactus spikes”

Saharia et al., “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, arXiv 2022.

What goes to Imagen?

Data

- Image-text pairs
- LAION-400M
- Internal (~500M images)

Sampler

- Classifier-free guidance
- Maximizing text-alignment

Model

- Diffusion models
- Cascading super-res
- Frozen text encoders

Scaling up

Saharia et al., “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, arXiv 2022.

What goes to Imagen?

Data

- Image-text pairs
- LAION-400M
- Internal (~500M images)

Sampler

- Classifier-free guidance
- Maximizing text-alignment

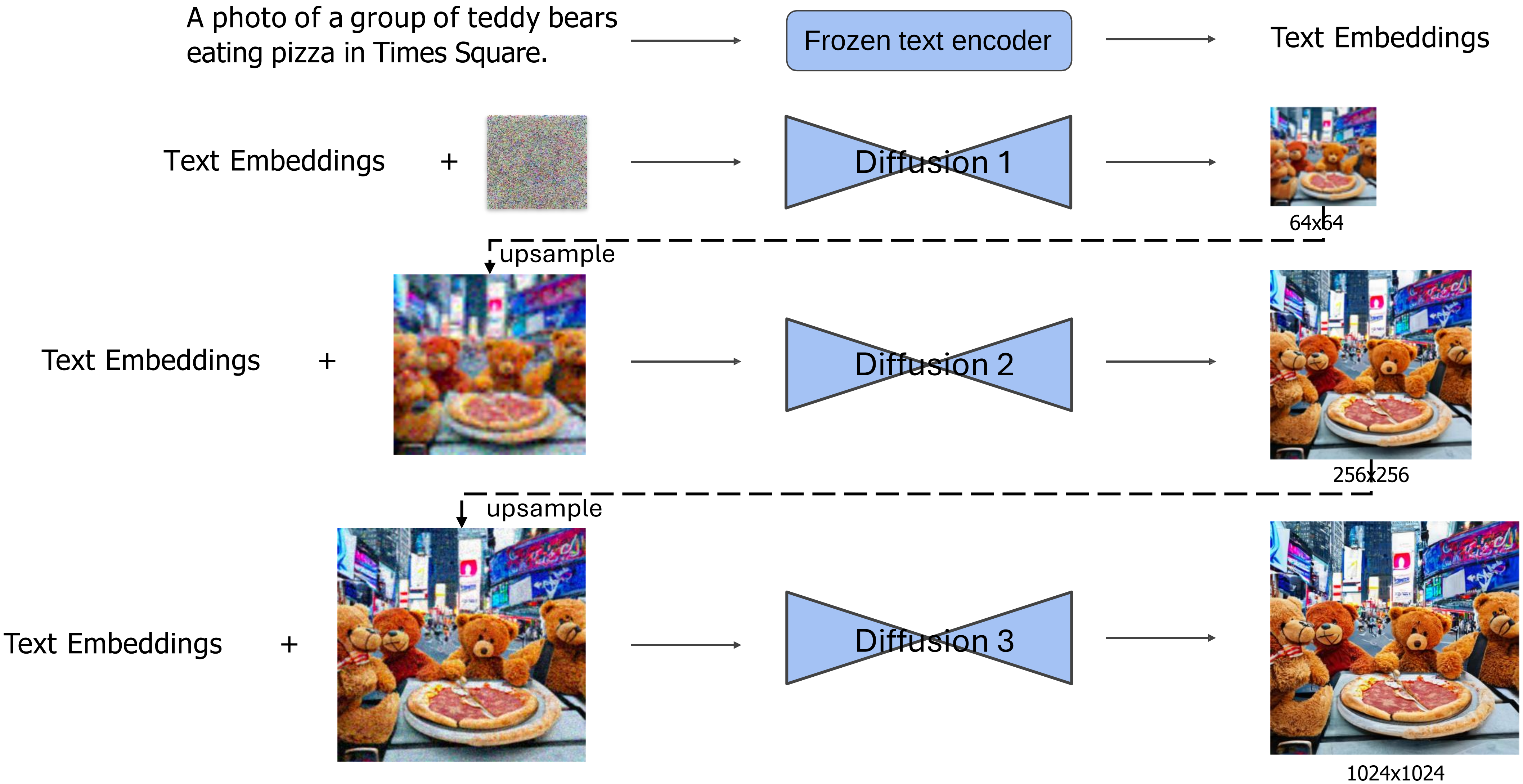
Model

- Diffusion models
- Cascading super-res
- Frozen text encoders

Scaling up

Saharia et al., “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, arXiv 2022.

Imagen: Cascaded generation pipeline



A photo of a group of teddy bears eating pizza in Times Square.

Frozen text encoder

Text Embeddings

Text Embeddings +

Diffusion 1

64x64

upsample

Text Embeddings +

Diffusion 2

256x256

upsample

Text Embeddings +

Diffusion 3

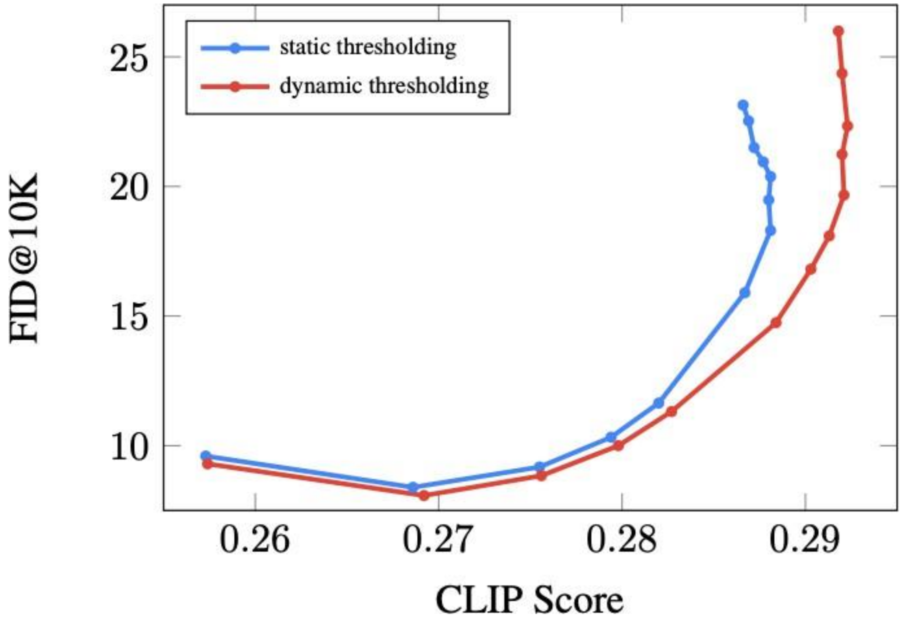
1024x1024

Saharia et al., “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, arXiv 2022.

Classifier-free guidance in Imagen

- Large classifier-free guidance weights → better text alignment, worse image fidelity

Better sample quality

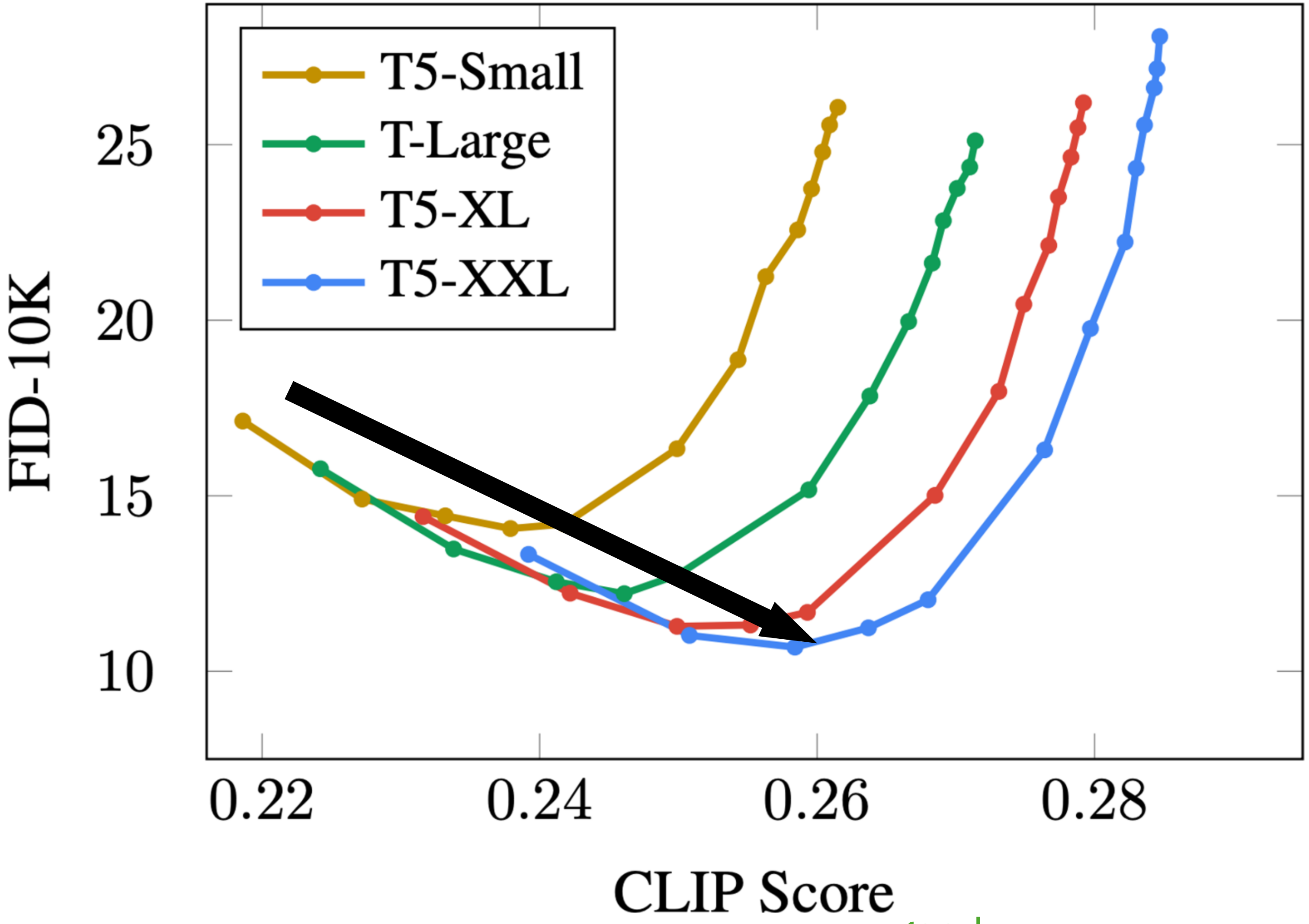


Better text alignment

More weights on classifier-free guidance for condition

Saharia et al., "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", arXiv 2022.

Larger Text Encoders → Better Alignment, Better Fidelity



bad

good

bad

good

Saharia et al., “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, arXiv 2022.

Can we generalize this to video?

Imagen Video

Imagen Video

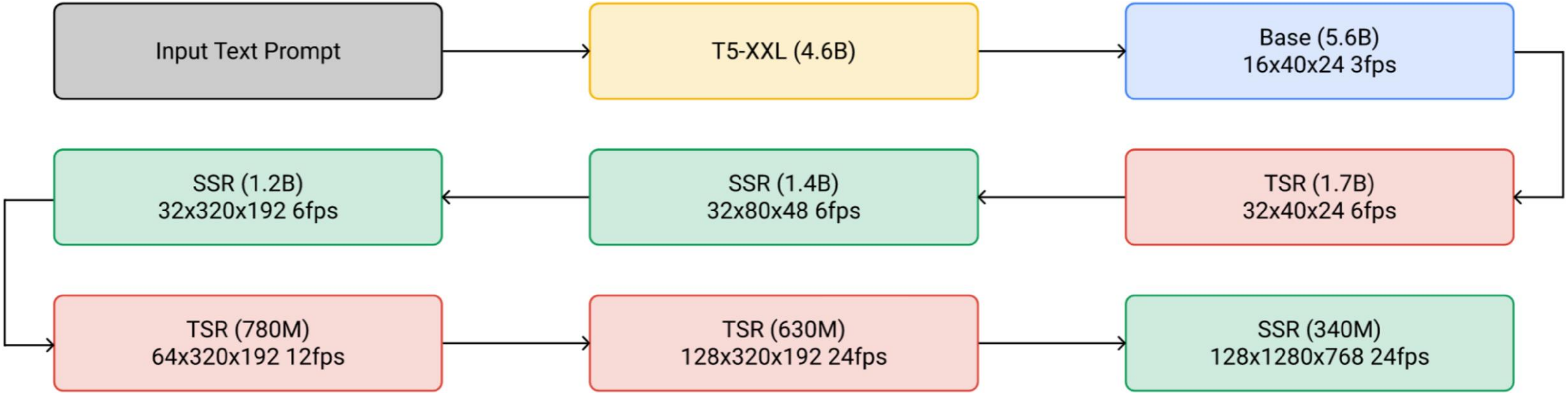
Imagen Video

Imagen Video

Imagen Video

Jonathan et al., “Imagen Video: High Definition Video Generation with Diffusion Models”, 2023.

Imagen Video generalizes **Imagen** to the video domain using a cascade of super-resolution diffusion models in space and time

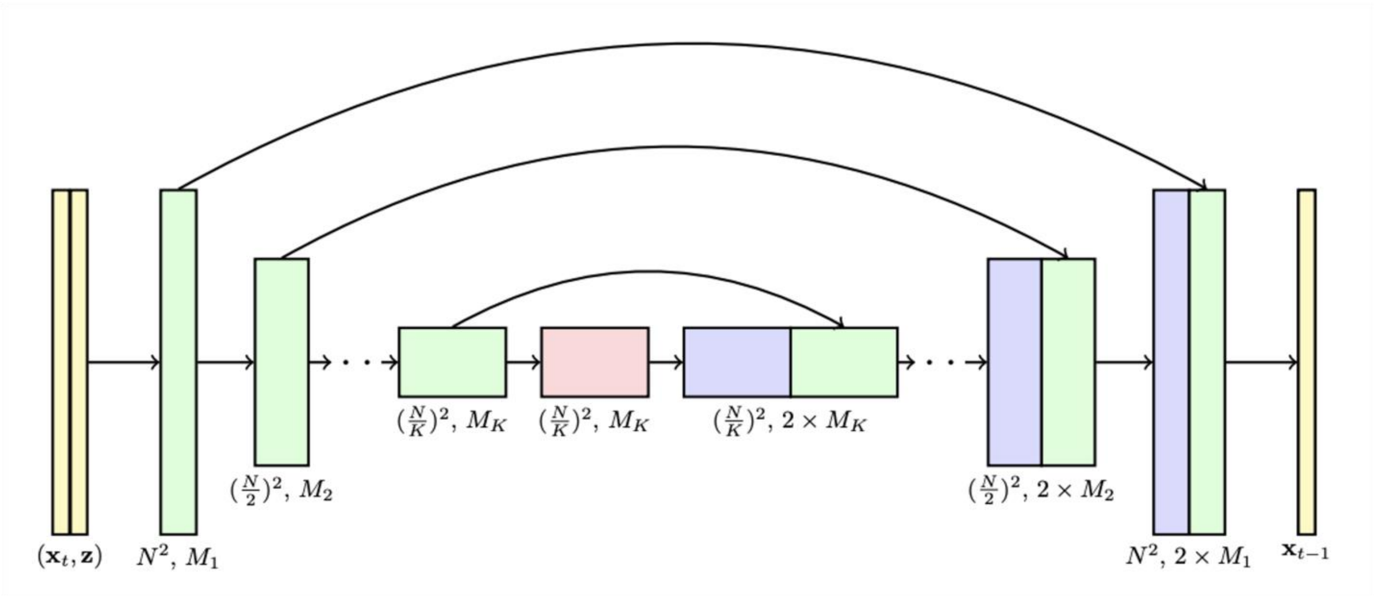


TSR: Temporal Super Resolution
SSR: Spatial Super Resolution
Output shape: frames × width × height
fps: frames per second

Video diffusion models

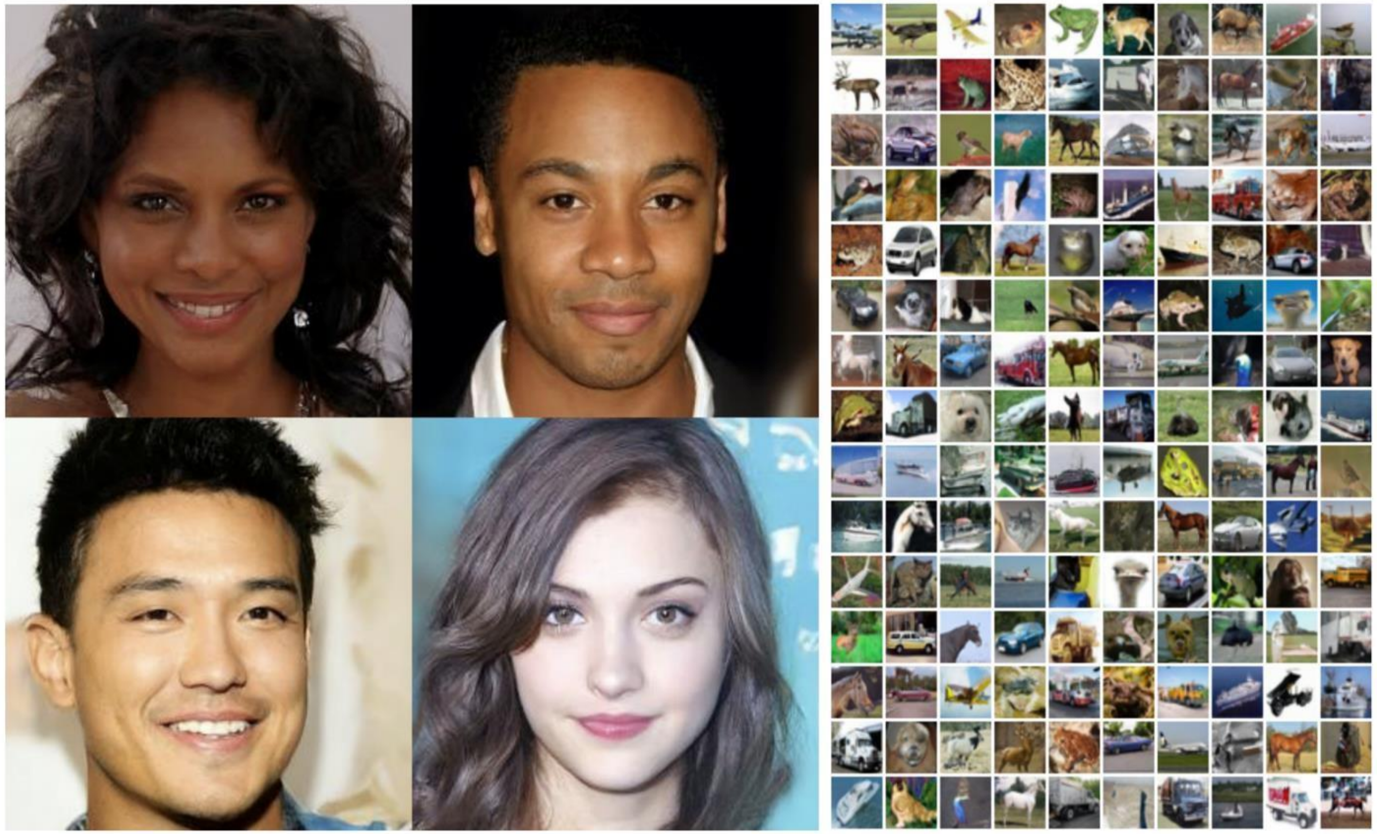
Ho & Salimans, et al. https://video-diffusion.github.io/

- Image → Video: Just add another dimension to the data tensor
- Image architecture: 2D UNet
- Video architecture: 3D UNet, space-time separable
  - repeat the 2D UNet over frames
  - additional layers to mix over time using attention or convolution



DDPM

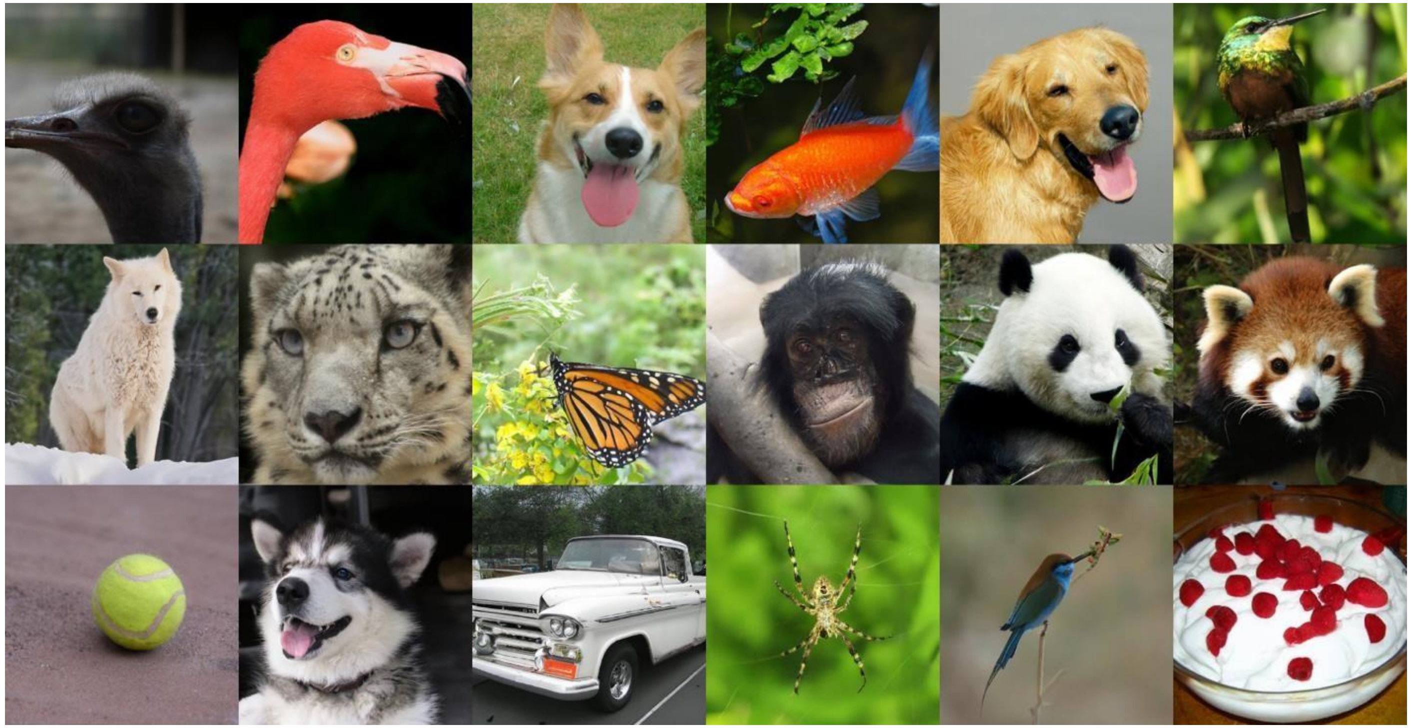
- Training diffusion models on raw images with a U-Net model



Ho et al. Denoising Diffusion Probabilistic Models. 2020.

Diffusion Models Beat GANs

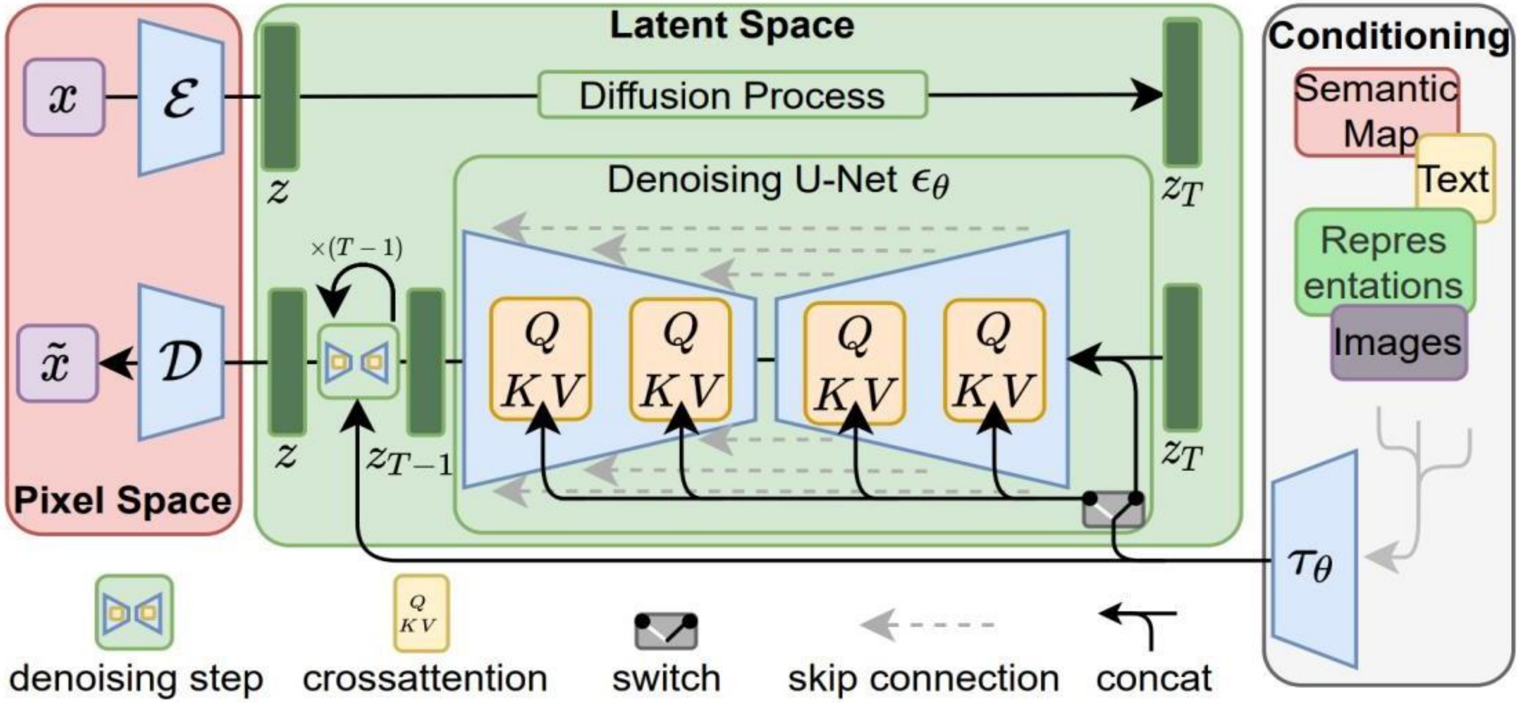
- Larger denoising model with sophisticated design
  - Adaptive group normalization
  - Attention layers in U-Net



Dhariwal et al. Diffusion Models Beat GANs on Image Synthesis. 2021.

Latent Diffusion Models (LDMs)

- Learn diffusion on VAE's latent
  - Yet another VAE! Except pre-trained.



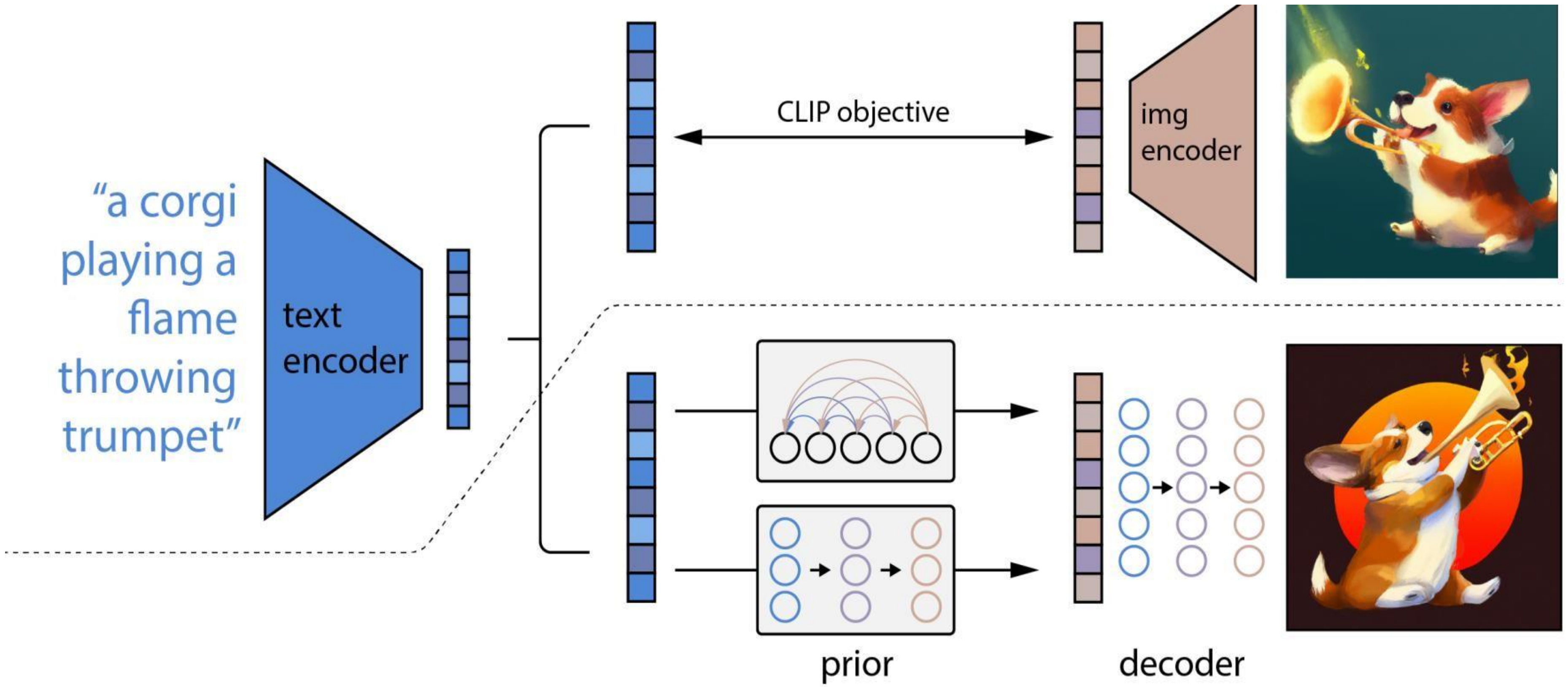
Rombach et al. High-Resolution Image Synthesis with Latent Diffusion Models. 2022.

Stable Diffusion

- Large-scale text-conditional LDMs
  - With VAEs trained also on larger datasets

Stability AI. https://github.com/Stability-AI/stablediffusion

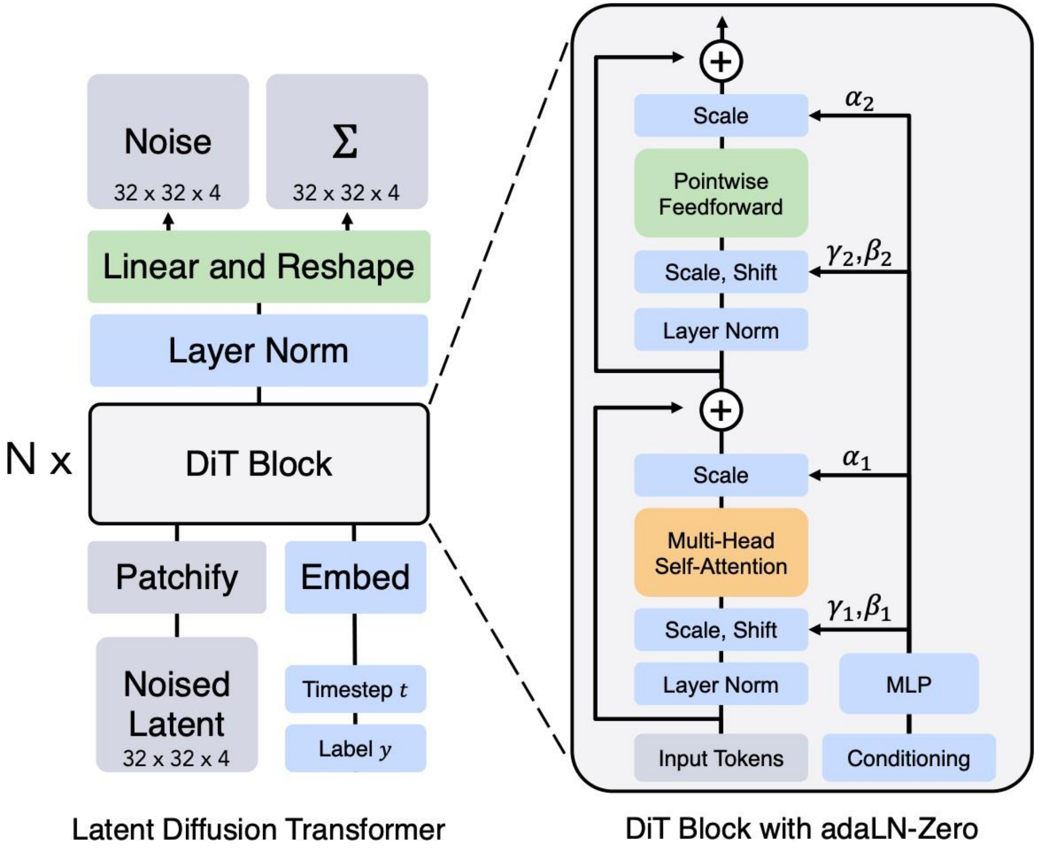
DALLE



Ramesh et al. Hierarchical Text-Conditional Image Generation with CLIP Latents

DiT

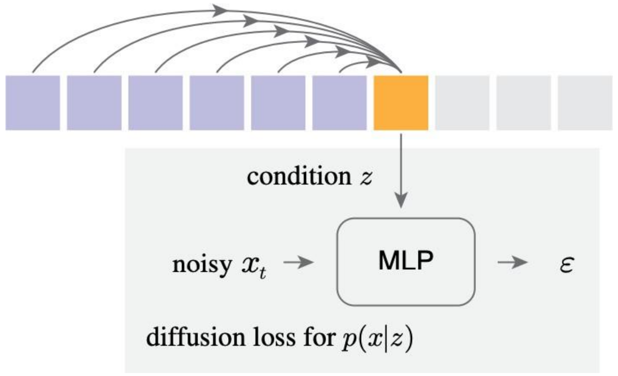
- A transformer architecture for diffusion models

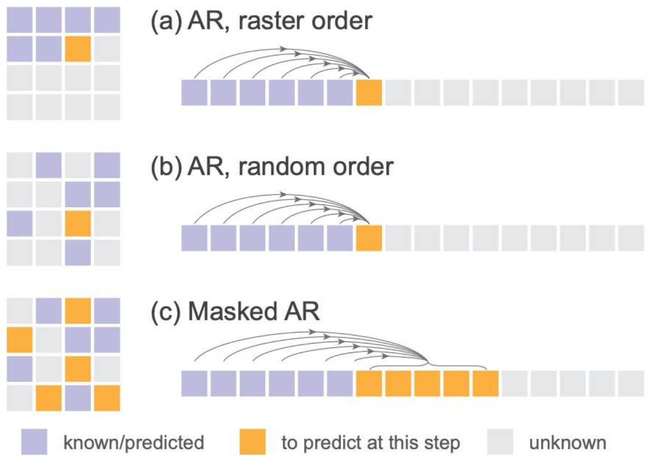


Peebles et al. Scalable Diffusion Models with Transformers. 2020.

MAR

- An autoregressive model with diffusion loss





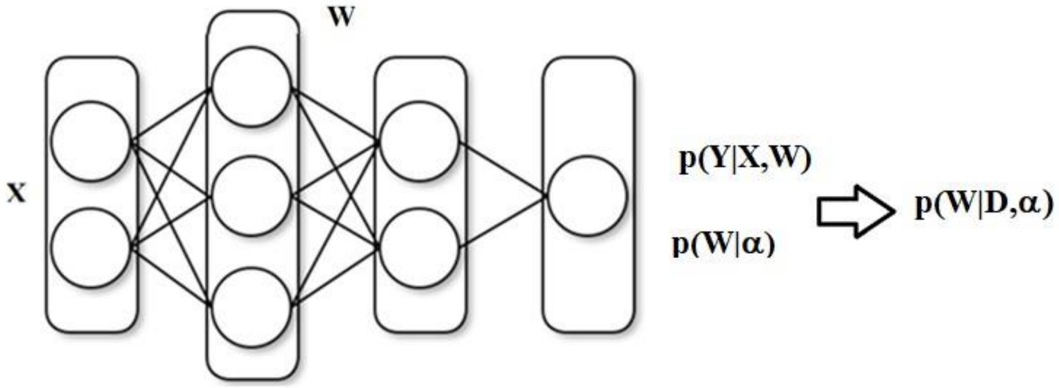
Li et al. Autoregressive Image Generation without Vector Quantization. 2024.

Bayesian Deep Neural Networks (BDNNs)

- BDNN Model Specification
- BDNN Inference Definition
- Understanding the Uncertainty and Its Applications
- BDNN Techniques
    - Bayesian Inference Approximation methods
        - Sampling
        - Variational estimation
    - Non-Bayesian Uncertainty Estimation Methods

Bayesian Deep Neural Networks (BDNNs)

- It extends the probabilistic deep models by capturing both data and model uncertainties
- Instead of estimating the model parameters, BDNN constructs the posterior distribution of the parameters p(**W** |**D**, α) and use it to perform prediction.
- BDNN consists of a likelihood model p(**y**|**x**, **W**) that relates the output **y** to the input **x,** and a prior model p(**W** |α) for the model parameters **W**



Prior of **W** Likelihood of **W**

Posterior of **W**

$$p(\mathbf{W} \mid D, \boldsymbol{\alpha}) = \frac{p(\mathbf{W} \mid \boldsymbol{\alpha})\, p(\mathbf{D} \mid \mathbf{W})}{\int p(\mathbf{W} \mid \boldsymbol{\alpha})\, p(\mathbf{D} \mid \mathbf{W}) d\mathbf{W}}$$

BDNN Model Specification

The parameter likelihood function p(**Y**|**X**, **W**) can be specified by a probabilistic DNN

- For regression problem
  - Let **X**∋ $\boldsymbol{R}^N$ be input vector, **Y** ∋ $\boldsymbol{R}^K$ be the output vector,

$$p(\mathbf{Y} \mid \mathbf{X}, \mathbf{W}) = N(\mathbf{Y}; \mu(\mathbf{X}, \mathbf{W}), \Sigma(\mathbf{X}, \mathbf{W}))$$

where $\mu()$ and $\Sigma()$ specify the mean and covariance matrix p(**Y**|**X**).

- For classification
  - Let **X**∋ $\boldsymbol{R}^N$ be the input vector and **Y** ∋ $\{1,2,\ldots,K\}$ be the output vector , and $\sigma_m()$ is the softmax function

$$p(Y \mid \mathbf{X}, \mathbf{W}) = Cat(Y \mid K, \boldsymbol{\lambda}(\mathbf{X}, \mathbf{W})), \qquad \boldsymbol{\lambda}(\mathbf{X}, \mathbf{W}) = \sigma_{\mathrm{M}}(\mathrm{f}(\mathbf{X}, \mathbf{W}))$$

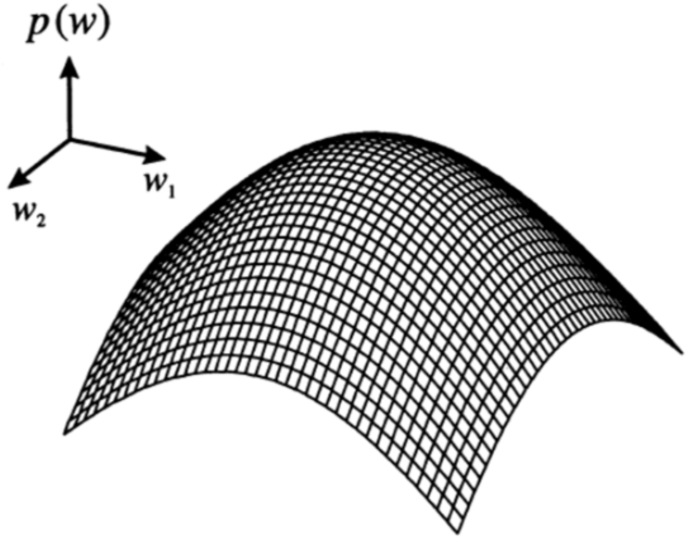
BDNN Model Specification (cont'd)

The prior probability distribution of the parameter p(**W**|**α**) can be specified as multi-variate Gaussian distribution, i.e.,

$$p(\mathbf{W} \mid \boldsymbol{\alpha}) = N(0, \Sigma) \approx N(0, \sigma^2 \mathbf{I})$$

I is a KxK identity matrix, K is dimension of **W**,

and σ^2 is a small positive number (e.g., 0.01)



BDNN Model specification (cont’d)

The posterior probability distribution of the parameter p(W |**D**, α) can be specified as follows

Given trainingdata $\mathbf{D} = (\mathbf{X}_i, \mathbf{Y}_i), i = 1,2,..,N$

$$p(\mathbf{W} \mid \mathbf{D}, \alpha) = \frac{p(\mathbf{W} \mid \alpha)\, p(\mathbf{D} \mid \mathbf{W})}{\int p(\mathbf{W} \mid \alpha)\, p(\mathbf{D} \mid \mathbf{W}) d\mathbf{W}}$$

$$= \frac{p(\mathbf{W} \mid \alpha) \prod_i p(\mathbf{Y}_i \mid \mathbf{X}_i, \mathbf{W})}{\int p(\mathbf{W} \mid \alpha) \prod_i p(\mathbf{Y}_i \mid \mathbf{X}_i, \mathbf{W}) d\mathbf{W}}$$

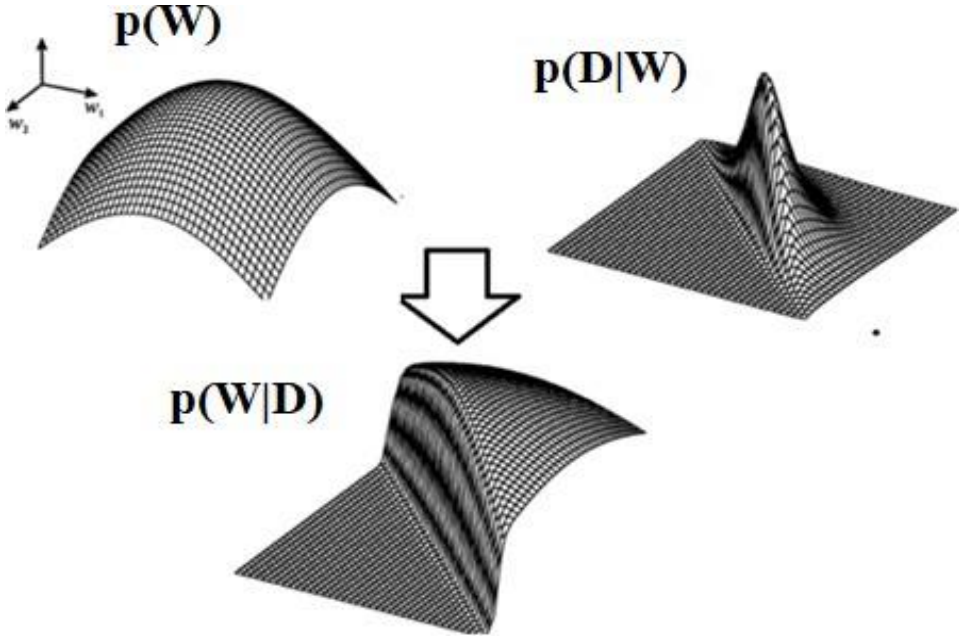


Figure credit: Aaron Courville

BDNN Inference Definition

- Empirical inference

$$\mathbf{Y}^* = \arg\max_{\mathbf{Y}} p(\mathbf{Y} \mid \mathbf{X}, \mathbf{D}, \boldsymbol{\alpha}^*) = \arg\max_{\mathbf{Y}} \int_{\boldsymbol{\Theta}} p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\Theta})\, p(\boldsymbol{\Theta} \mid \mathbf{D}, \boldsymbol{\alpha}^*) d\boldsymbol{\Theta}$$

- Full Bayesian inference

$$\mathbf{Y}^* = \operatorname*{argmax}_{\mathbf{Y}} p(\mathbf{Y} \mid \mathbf{X}, \mathbf{D}) = \operatorname*{argmax}_{\mathbf{Y}} \int_{\boldsymbol{\Theta}} \int_{\boldsymbol{\alpha}} p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\Theta})\, p(\boldsymbol{\Theta} \mid \mathbf{D}, \boldsymbol{\alpha})\, p(\boldsymbol{\alpha} \mid \mathbf{D}) d\boldsymbol{\Theta} d\boldsymbol{\alpha}$$

Empirical Bayesian Inference

Empirical BDNN inference is to predict output **Y** given **X**, **D**, and **α**, where **α** is either given or learnt from data **D** (denoted as α^*)

$$Y^* = \arg\max_{Y}\, p(Y \mid X, \mathbf{D}, \boldsymbol{\alpha}^*)$$

where $$p(Y \mid X, D, \boldsymbol{\alpha}^*) = \int_{\mathbf{W}} p(Y \mid X, \mathbf{W})\, p(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\alpha}^*)\, d\mathbf{W} = E_{p(\mathbf{W}|\mathbf{D},\boldsymbol{\alpha}^*)}(p(Y \mid X, \mathbf{W}))$$

$$\boxed{Var(Y \mid X, \mathbf{D}, \alpha)} = E_{p(Y|X,\mathbf{D},\alpha)}(Y^2 \mid X, \mathbf{D}, \alpha) - E_{p(Y|X,\mathbf{D},\alpha)}{}^{2}(Y \mid X, \mathbf{D}, \alpha)$$

$$= \underbrace{\boxed{E_{p(\mathbf{W}|\mathbf{D},\alpha)}[Var_{p(\mathbf{Y}|\mathbf{X},\mathbf{W})}(Y \mid X, \mathbf{W})]}}_{\text{Data (aleatoric) uncertainty}} + \underbrace{\boxed{Var_{p(\mathbf{W}|\mathbf{D},\alpha)}[E_{p(\mathbf{Y}|\mathbf{X},\mathbf{W})}(Y \mid X, \mathbf{W})]}}_{\text{Model (epistemic) uncertainty}}$$

Total uncertainty
in variance

Full Bayesian Inference

Given **X** and training data **D**, full Bayesian inference is to predict output **Y** by

$$\mathbf{Y}^* = \underset{\mathbf{Y}}{\operatorname{argmax}}\ p(\mathbf{Y} \mid \mathbf{X}, \mathbf{D})$$

where $p(\mathbf{Y} \mid \mathbf{X}, \mathbf{D}) = \iint\limits_{\mathbf{W}\boldsymbol{\alpha}} p(\mathbf{Y}, \mathbf{W}, \boldsymbol{\alpha} \mid \mathbf{X}, \mathbf{D}) d\mathbf{W} d\boldsymbol{\alpha}$

$$= \iint\limits_{\mathbf{W}\boldsymbol{\alpha}} p(Y \mid \mathbf{X}, \mathbf{W})\, p(\mathbf{W}, \boldsymbol{\alpha} \mid \mathbf{D}) d\mathbf{W} d\boldsymbol{\alpha} = E_{p(\mathbf{W}, \boldsymbol{\alpha} \mid \mathbf{D})}(p(Y \mid \mathbf{X}, \mathbf{W}))$$

$$= \iint\limits_{\mathbf{W}\boldsymbol{\alpha}} p(Y \mid \mathbf{X}, \mathbf{W})\, p(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\alpha})\, p(\boldsymbol{\alpha} \mid \mathbf{D}) d\mathbf{W} d\boldsymbol{\alpha} = E_{p(\boldsymbol{\alpha} \mid \mathbf{D})}(E_{p(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\alpha})}(p(Y \mid \mathbf{X}, \mathbf{W})))$$

$$Var(\mathbf{Y} \mid \mathbf{X}, \mathbf{D}) = E(\mathbf{Y}^2 \mid \mathbf{X}, \mathbf{D}) - E^2(\mathbf{Y} \mid \mathbf{X}, \mathbf{D})$$

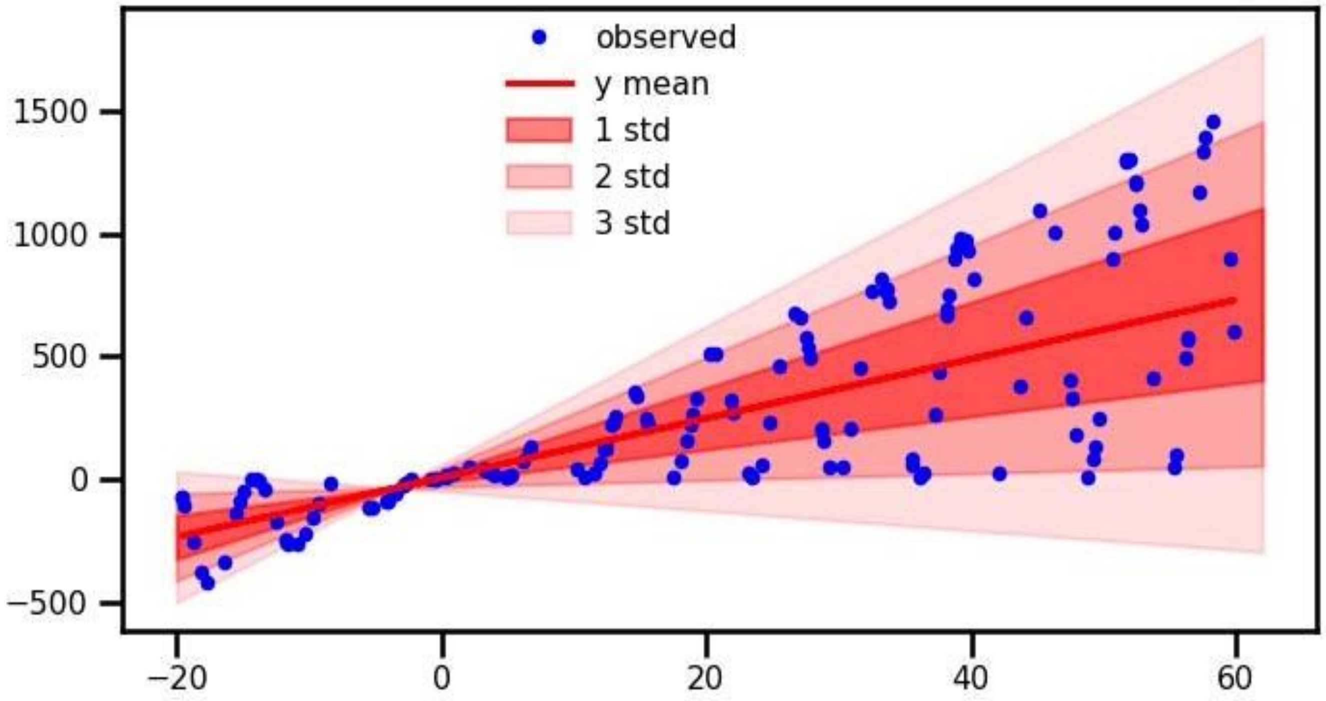
$$= E_{p(\boldsymbol{\alpha} \mid \mathbf{D})}(Var(\mathbf{Y} \mid \mathbf{X}, \mathbf{D}, \boldsymbol{\alpha})) + Var_{p(\boldsymbol{\alpha} \mid D)}(E_{p(\mathbf{Y} \mid \mathbf{X}, \mathbf{D}, \boldsymbol{\alpha})}(\mathbf{Y} \mid \mathbf{X}, \mathbf{D}, \boldsymbol{\alpha}))$$

Expected total uncertainty
(total aleatoric uncertainty)

Uncertainty of total expectation
(total epistemic uncertainty)

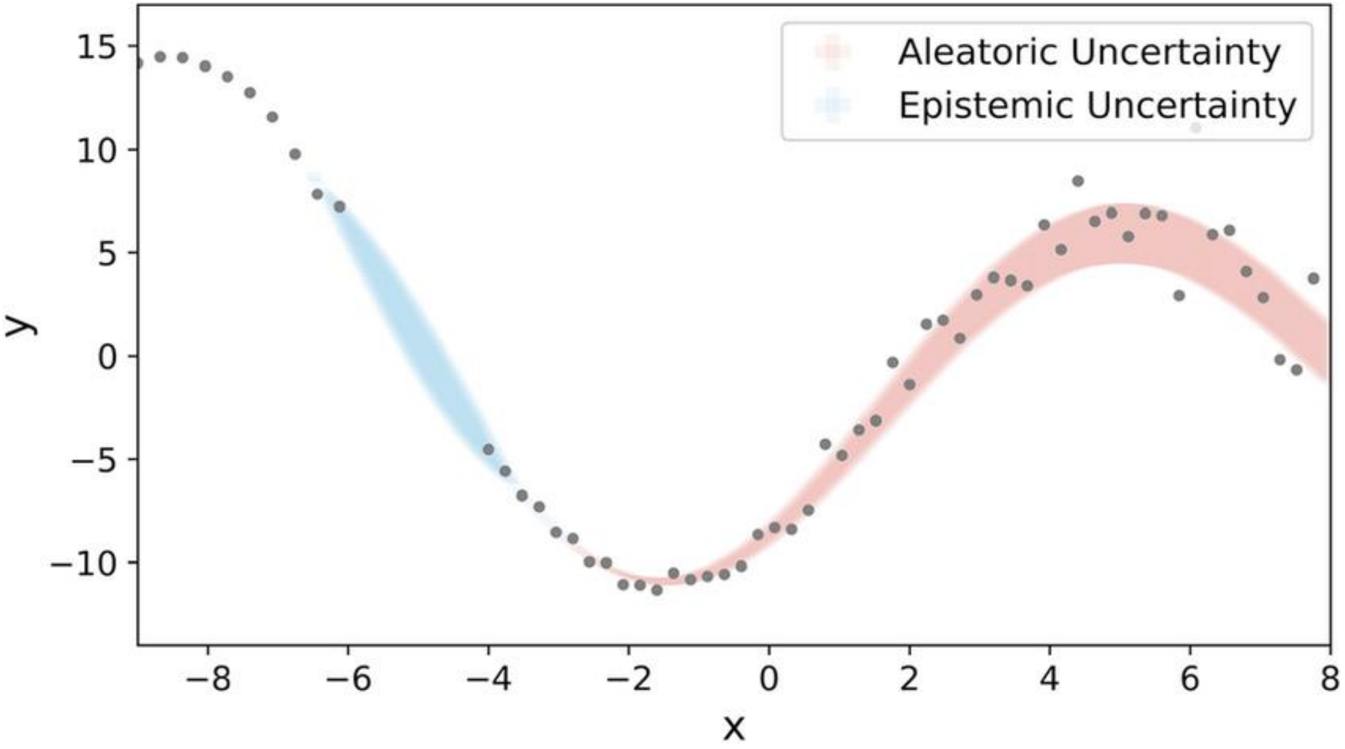
Variance from Bayesian Neural Networks

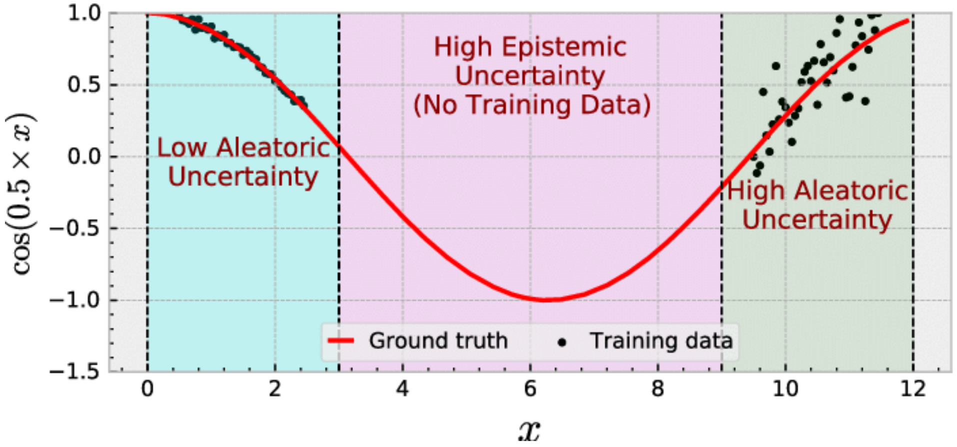
Aleatoric (Data) Uncertainty



- Here, the data gets more spread further away from the origin, so it makes sense that the variance also varies

Epistemic vs Aleatoric Uncertainty





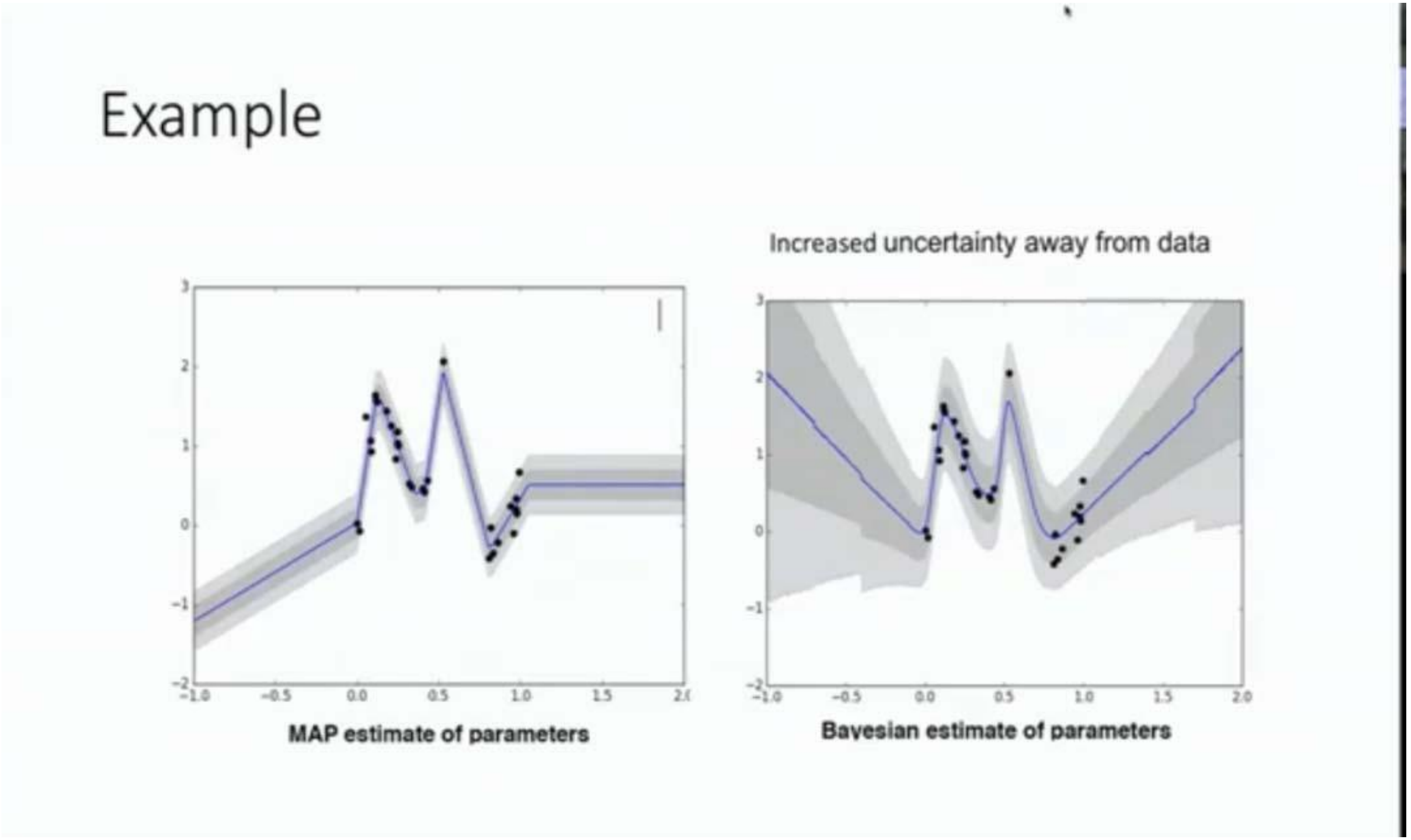
- We expect the epistemic uncertainty to be large in regions where the training data is sparse, and aleatoric in regions of large noise

Epistemic vs Aleatoric Uncertainty

- **Aleatoric (data) uncertainty** is about irreducible randomness in the observation process
    - Irreducible given the model

- **Model (epistemic) uncertainty** refers to the uncertainty about the model parameters (e.g., weights W or hyperparameters α).
    - It reflects how unsure we are about which model (or function) is correct.
    - Caused by lack of knowledge or insufficient training data.
    - Can be reduced by collecting more data.

MAP Estimate v.s. Bayesian estimate



Example

Increased uncertainty away from data

MAP estimate of parameters

Bayesian estimate of parameters

From Max Welling youtube talk

Uncertainty with Benchmark Datasets

MNIST	LeNet5	VGG16	ResNet
Accuracy	0.977	0.985	**0.992**
Total	0.07269	0.07235	**0.07221**
Aleatoric	**0.07076**	0.07105	0.07102
Epistemic	0.00193	0.00130	**0.00119**

CIFAR10	LeNet5	VGG16	ResNet
Accuracy	0.919	0.974	**0.980**
Total	0.08837	0.08853	**0.08579**
Aleatoric	0.07466	**0.07138**	0.07899
Epistemic	0.01380	0.01715	**0.00680**

Larger model leads to smaller total uncertainty, mainly because of smaller epistemic uncertainty !
CIFAR has larger total uncertainty over MNIST dataset.

Applications of Epistemic Uncertainty

- Model selection
- Model fusion
- Active learning (select the samples with large epistemic uncertainty)
- Outlier or anomaly (out of distribution) detection (detect input with large epistemic uncertainty)
- Unbalance data bias correction
- Model adaptation/generalization/domain adaptation

Bayesian Model Selection

- BNN can be used to select best NN models (M) for a given training data **D** in terms of its architecture (number of hidden layers, number of filters, types of activation functions, etc..) using marginal model likelihood, i.e.,

$$p(\mathbf{D} \mid M) = \int_{\mathbf{W}} \underbrace{p(\mathbf{D} \mid \mathbf{W}, M)}_{\text{Model likelihood}} \underbrace{p(\mathbf{W} \mid M)}_{\text{model prior}} d\mathbf{W}$$

$$M^* = \underset{M}{\operatorname{argmax}}\ p(\mathbf{D} \mid M)$$

The prior term balances between data likelihood term to avoid overly complex models.

Model Fusion for Prediction

- **Model fusion**
  - Given K models M_k , k=1,2,..K, training data **D**, and an input X, we can also combine the results from the K models, i.e.

$$p(Y \mid X, D) = \sum_{i=1..K} p(Y \mid X, D, M_i)\, p(M_i \mid D)$$

$$y^* = \text{argmax } p(y \mid X, D)$$

The result is a weighted average of the probability distributions over the outputs of the models.

Bayesian Deep Neural Networks (BDNNs)

- BDNN Model Specification
- BDNN Inference Definition
- Understanding the Uncertainty and Its Applications
- BDNN Techniques
    - Bayesian Inference Approximation methods
        - Sampling
        - Variational estimation
    - Non-Bayesian Uncertainty Estimation Methods

Parameter Integration Approximation

- Both empirical and full Bayesian inference require computing expectation over the parameters $\mathbf{w}$.
- Such parameter expectation becomes intractable as it requires integration over a large number of parameters.
- Approximations are often used to approximate parameter integration
  - Monte Carlo methods - approximate expectation with sample mean
  - Variational methods - approximate posterior with simple distribution

The Monte Carlo Methods

Expected value of a function: $\mathrm{E}[f(x)] = \int p(x)f(x)dx$

can be approximated by sample averages

$$\hat{s} = \frac{1}{N}\sum_{n=1}^{N} f(x_n)$$

where x_n~ p(x)

The sample average $\hat{s}$ is an unbiased estimate of E[f(x)] and as N approaches to infinite, the sample average $\hat{s}$ approaches E[f(x)].

Sampling method

- Empirical Bayesian inference

$$Y^* = \arg\max_{Y} p(Y \mid X, \mathbf{D}, a)$$

$$= \underset{Y}{\text{argmax}} \int_{\mathbf{W}} p(Y \mid X, \mathbf{W})\, p(\mathbf{W} \mid \mathbf{D}, a)$$

$$\approx \arg\max_{Y} \frac{1}{N} \sum_{n=1}^{N} p(Y \mid X, \mathbf{W}_n), \text{ where } \mathbf{W}_n \sim \text{ p}(\mathbf{W} \mid \mathbf{D}, a)$$

$$= \frac{1}{N} \sum_{n=1}^{N} \arg\max_{Y_n^*} p(Y_n, \mid X, \mathbf{W}_n)*$$

* Changing the order may not hold for categorical RVs

Sampling method (cont’d)

- Full Bayesian Inference

$$\mathbf{Y}^* = \arg\max_{Y} p(\mathbf{Y} \mid \mathbf{X}, \mathbf{D})$$

$$= \arg\max_{Y} \int_{\mathbf{W}}\int_{\boldsymbol{\alpha}} p(\mathbf{Y} \mid \mathbf{X}, \mathbf{W}) p(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\alpha}) p(\boldsymbol{\alpha} \mid \mathbf{D}) d\mathbf{W} d\boldsymbol{\alpha} = E_{p(\boldsymbol{\alpha}|\mathbf{D})}(E_{p(\mathbf{W}|\mathbf{D},\boldsymbol{\alpha})}(\mathbf{Y} \mid \mathbf{X}, \mathbf{W})))$$

$$\approx \arg\max_{Y} \frac{1}{N_\alpha N_{\mathbf{W}}} \sum_{n_\alpha=1}^{N_\alpha} \sum_{n_{\mathbf{W}}=1}^{N_{\mathbf{W}}} p(Y \mid \mathbf{W}_{n_w n_\alpha}, X), \text{ where } \boldsymbol{\alpha}_{n_\alpha} \sim p(\boldsymbol{\alpha} \mid \mathbf{D}), \mathbf{W}_{n_w n_\alpha} \sim \mathrm{p}(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\alpha}_{n_\alpha})$$

$$= \frac{1}{N_\alpha N_w} \sum_{n_\alpha=1}^{N_\alpha} \sum_{n_w=1}^{N_w} \arg\max_{Y} p(Y \mid \mathbf{W}_{n_w n_\alpha}, X)\,*$$

* May not be true for categorical RVs

Sampling Challenge

$$\mathbf{W}_s \sim p(\mathbf{W} \mid \mathbf{D}, \alpha) = \frac{p(\mathbf{W} \mid \alpha)\prod_i p(\mathbf{Y}_i \mid \mathbf{X}_i, \mathbf{W})}{\int p(\mathbf{W} \mid \alpha)\prod_i p(\mathbf{Y}_i \mid \mathbf{X}_i, \mathbf{W})d\mathbf{W}}$$

- **W** is high dimensional, with millions or billons of parameters
- The denominator requires integration over all W , which is intractable.

Efficient Sampling Methods

- Importance sampling

- **Markov Chain Monto Carlo (MCMC) Sampling**
  - Gibbs sampling
  - Metropolis hastings sampling
  - Hamiltonian Monto Carlo sampling

Markov Chain Monte Carlo (MCMC)

- One of the most powerful methods for sampling in **high dimensional parameter** space.

- Starting from a random initialization, MCMC sampling follows a Markov chain to generate a sequence of samples $x^{(1)}, \ldots, x^{(n)}$ from the distribution $p(x)$, where each sample $x^{(i)}$ depends (only) on the previous sample $x^{(i-1)}$.

- For an ergodic an aperiodic markov chain, independent of starting point, as $n \rightarrow \infty$, sample distribution will approach $p(x)$.

Gibbs Sampling

- One of the simplest and most popular MCMC sampling methods
- It samples one parameter x_i at a time, given the current values of other parameters, i.e., sample x_i from $p(x_i | x_{-i})$ where x_{-i} are the remaining parameters except for x_i.
- At each step, it needs compute $p(x_i | x_{-i})$ from $p(\mathrm{x})$.

Metropolis Hastings Sampling

- Gibbs sampling assumes the availability of $p(x_i|x_{-i})$ and its efficient estimation.

- This is NOT possible for BDNNs as it includes a normalization constant that requires integration over all **W**.

- An alternative MCMC method is Metropolis-Hastings (M-H) sampling method.

- As a rejection sampling method, M-H samples from a simpler proposal distribution q(x) to choose the next $x^{(j)}$, and then decide if to accept the sample using the unnormalized $\bar{p}$(x) (avoiding the parameter integral).

Metropolis Hastings Sampling

Metropolis-Hastings is very powerful and widely-used.

It reduces the problem of sampling from a difficult distribution $p(x)$ to **making proposals**: $q(x^{(\mathrm{j})} \mid x^{(\mathrm{j}-1)})$ and **evaluating ratios** of $p(x^{(\mathrm{j})})/\mathrm{p}(x^{(\mathrm{j}-1)})$.

The proposals can be trivial, e.g. random walk (choose $x^{(\mathrm{j})}$ from a normal distribution centered at $x^{(\mathrm{j}-1)}$), or sophisticated. Only **efficiency, not correctness** is affected by the proposal.

Because only ratios $p\left(x^{(\mathrm{j})}\right)/p\left(x^{(\mathrm{j}-1)}\right)$ are needed, **we don't have to deal with normalizing sums** Z.

Metropolis Hastings Sampling

1. Start with $\mathbf{x}^0 = \{x^0_1, x^0_2, \ldots, x^0_n\}$ for n variables, whose joint distribution is p(x).
2. At iteration t, obtain a new sample of $\mathbf{x}^t$ from $q(x^t \mid x^{t-1})$, where q is a symmetric proposal distribution.

   q() needs be a symmetric distribution and a Gaussian is often chosen.

3. Calculate

$$a = \frac{\overline{p}(x^t)}{\overline{p}(x^{t-1})}, \text{ where } \quad \overline{p}(x) \propto p(x)$$

4. Accept x^t with a probability of min(1, a). If x^t is rejected, $x^t = x^{t-1}$
5. Repeat steps 2-4 M times to get M samples, with some burn-in.

Hamiltonian Monte-Carlo

- M-H proposals use random walks, which are slow in exploration of large parameter space.

- Hamiltonian Monte-Carlo addresses this problem by treating sampling of posterior density as a problem in Hamiltonian dynamics, by adding an independent velocity variable (***v***) to the parameter space **W**, yielding the joint probability p(**W**,**v**).

- Sampling proposals can be generated analytically from p(**W**,**v**) via physics simulation following Hamiltonian dynamics, which leads to more samples from high probability (low energy) regions.

- Converge faster than M-H method but requires computing gradient each step.

Stochastic Gradient Hamiltonian Monte Carlo

The stochastic gradient Hamiltonian Monte Carlo (SGHMC) extends the HMC by replacing the potential gradient with it’s stochastic version, that is, computing the potential gradient from randomly sampled mini-batches from the data.

Sampling Methods Summary

- The goal is to approximate the parameter integration with sample average .
- Monte-Carlo sampling methods:
  - Gibbs sampling: simple but requires closed form local distributions
  - M-H sampling: uses a simple proposal distribution to generate next samples and use an unnormalized posterior to accept/reject samples. But it is slow.
  - Hamiltonian Monte Carlo (HMC) adds a momentum term to speed up the proposal generations.
  - Stochastic Hamiltonian Monte Carlo (HMC) computes gradient using mini-batch.

Variational methods

- Variational inference methods approximate the parameter integration by approximating the posterior parameter distribution p(**W**|**D**, **α**) with a simple and factorized distribution q(**W**|**D**, **β**).

$$p(\mathbf{W} \mid \mathbf{D}, \alpha) \approx q(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\beta})$$

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} KL(q(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\beta}) \,||\, p(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\alpha}))$$

where **β** are the variational parameters.

- With a factorized distribution, the integration and sampling can be performed more easily.

$$p(Y \mid X, D, \boldsymbol{\alpha}^*) = \int_{\mathbf{W}} p(Y \mid X, \mathbf{W})\, p(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\alpha}^*) d\mathbf{W} \approx \int_{\mathbf{W}} p(Y \mid X, \mathbf{W}) q(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\beta}) d\mathbf{W}$$

$$\approx \frac{1}{N} \sum_{s=1}^{N} p(Y \mid X, \mathbf{w}_s) \qquad \mathbf{w}_s \sim q(\mathbf{W} \mid \mathbf{D}, \boldsymbol{\beta})$$

Variational Methods

- Design choice of q(**W**|D)

- Often choose to be simple for tractable inference e.g. mean-field variational inference assumes fully factorized

$$q(\mathbf{W}|\mathbf{D}) = \prod_i \boldsymbol{p}(W_i | \boldsymbol{D})$$

  - Simple q() leads to large gap between p() and q().
  - More sophisticated structure or model can be used to define $q()$ to better reflect the posterior

- Representative variants of variational inference methods
  - Mean-field variational inference [Jordan 1999]
  - Stochastic variational inference [Graves 2011, Paisley 2012, Hoffman 2013, Ranganath 2014]
  - Inference networks [Minh 2014, Kingma 2014, Rezende 2014]

- Pros: Fast and easy to implement
- Cons: Inaccurate due to the inherent gap between q() and p(), leading to non-optimal solution. Requiring an optimization procedure to estimate q() that varies with initialization and could be slow for large model.

- Train deep generative models such as GANs or normalizing flows as the variational model to efficiently generate samples?
  - Samples of **W** !!!

MCMC methods v.s. Variational methods

- MCMC methods can be thought as exact methods as their estimation accuracy approaches the exact values asymptotically as the number of samples approaches to very large.
  - Sampling, however, is time consuming and cannot scale up well.
  - Furthermore, it is hard to know when the burning-in period ends.

- Variational methods are more efficient but they are always approximate as there is always a gap between the variational distribution and that of the true posterior distribution.
  - Moreover, it requires an optimization procedure to learn the variational parameters.

- Hybrid methods may be employed that combine the MCMC with the variatonal methods.
  - In addition, parameters can be divided into two sets, one set can be solved analytically, while another set can be solved by sampling or variaitonal method.

Other Bayesian Methods

- **Last Bayesian layer** - fix the parameters for all layers except for the last layer. Only need to construct the posterior for the parameters in the last layer, hence significantly reduce the number of parameters.

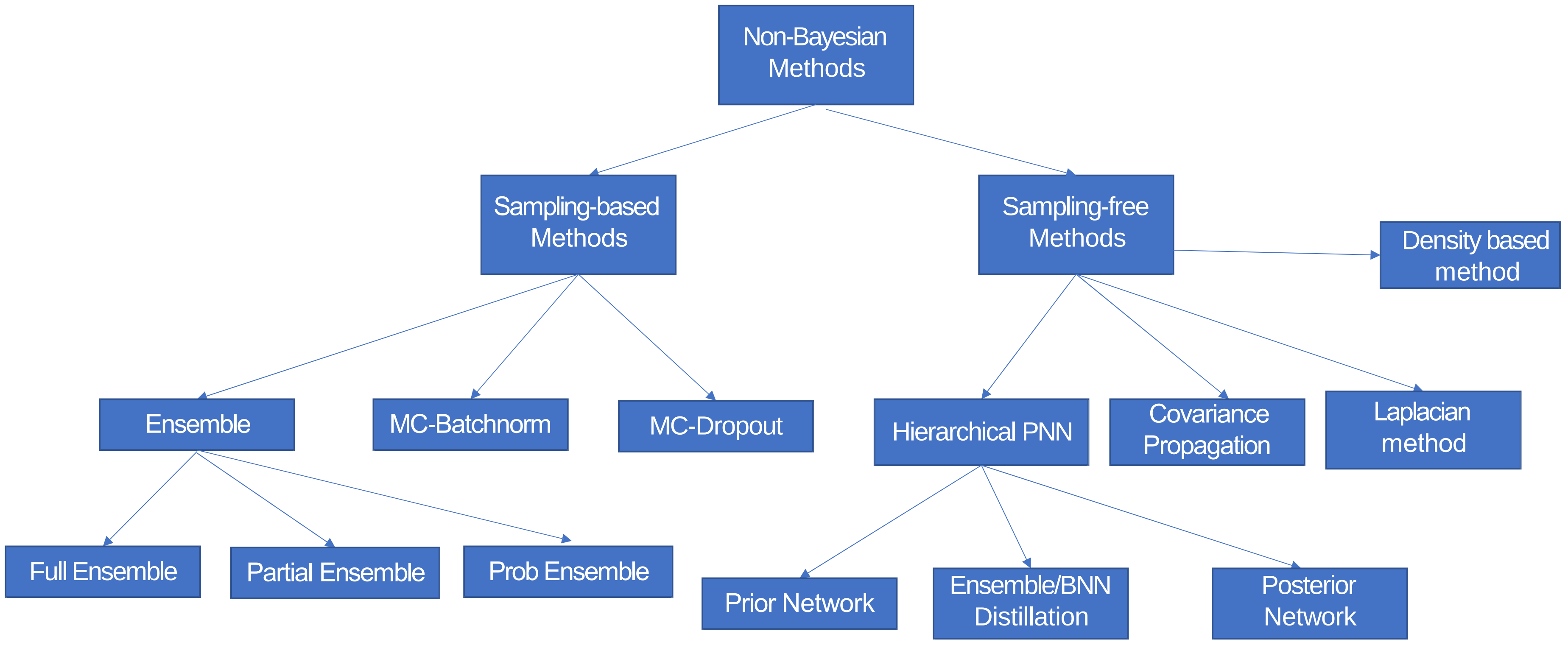
- **Attention-based methods**, where NN parameters are fixed but attention parameters are probabilistic .

- **Parameter projection**- project **W** into a lower dimensional space **Z** . Construct the posterior of Z and sample Z and then use the sampled Z to recover sample of W.

- **Deep generative models** may be used to produce samples of **W** from a simple latent distribution **Z**.

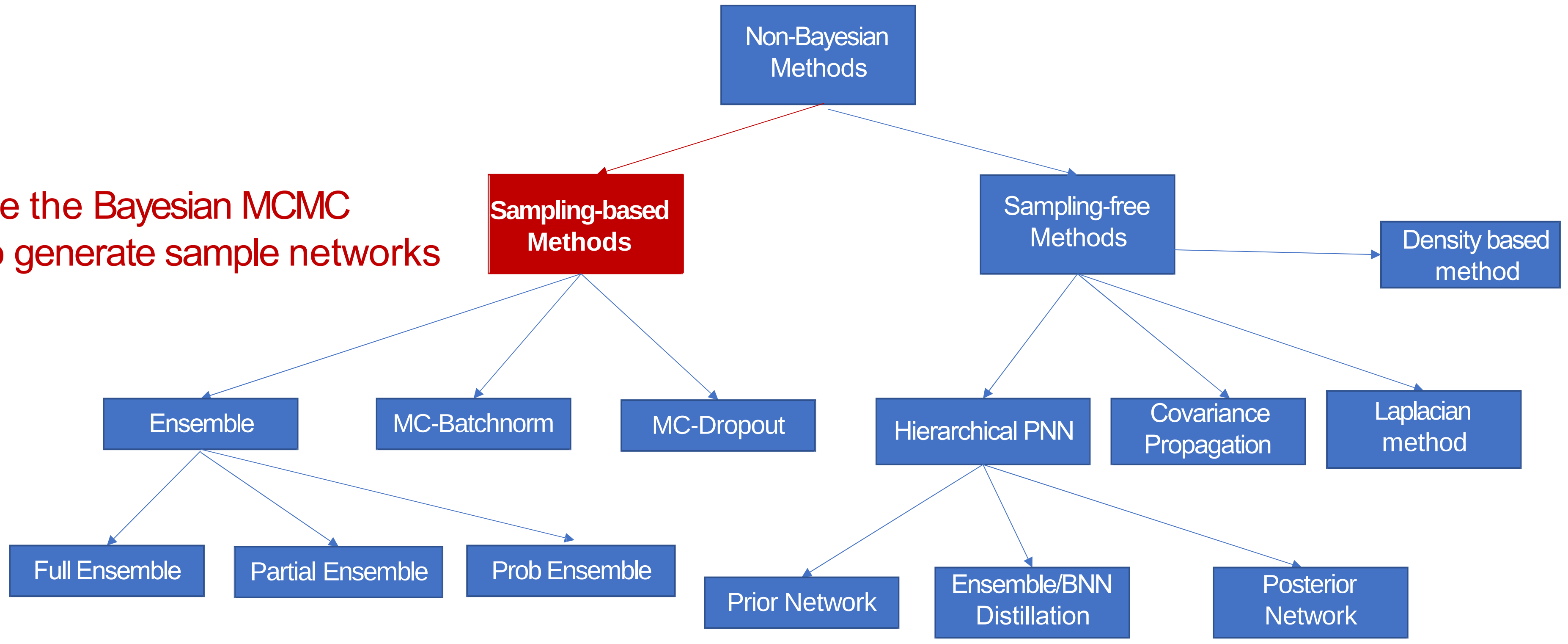
Non-Bayesian Uncertainty Estimation Methods

Types of Non-Bayesian Uncertainty Estimation Methods



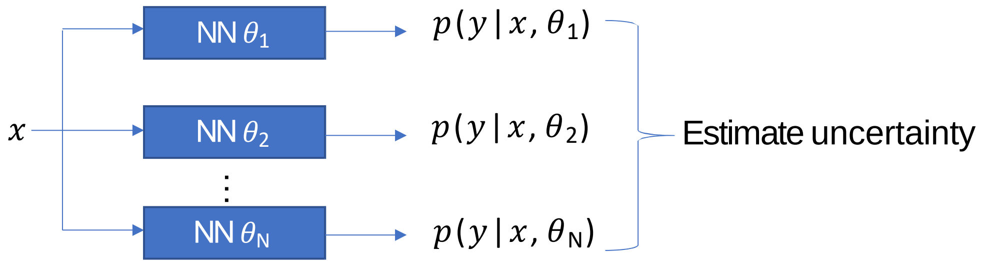
Types of Non-Bayesian Uncertainty Estimation Methods

approximate the Bayesian MCMC sampling to generate sample networks



Ensemble Methods

- Full Ensemble
  - Use different initializations to train multiple models (typically 5 to 10) to get multiple parameters of neural networks. Different parameters provide different outputs.
  - We have a bunch of output samples from ensemble models for uncertainty estimation

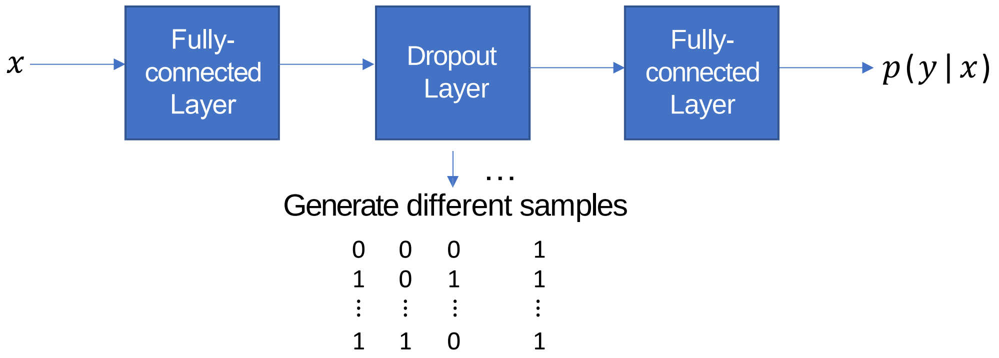


- Partial Ensemble
  - Partial Ensemble improves the ensemble method in terms of computational and memory costs. It reduces the number of parameters needed for constructing the whole ensemble model.
  - Training Partial Ensemble:
    - Step 1: pre-train the whole parameters $\theta = \{\theta_1, \theta_2\}$
    - Step 2: fix θ_1, randomly initialize θ_2^i , then train θ_2^i, $i = 1,2,\ldots, N$

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles." *arXiv preprint arXiv:1612.01474* (2016).

MC-Dropout

- Dropout is a regularization technique for conventional deep neural networks, where it follows the Bernoulli distribution to decide which node to keep or drop.
- Key idea: apply the dropout during testing to produce different networks as BNN samples



- Collect the inference results $p^{t}(\mathrm{y}\,|\,\mathrm{x})$ $(t = 1{,}2 \ldots, T)$ from each sample and average these results.
- It has been shown that drop-out samples are mathematically equivalent to BNN samples.

Gal, Yarin, and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." *international conference on machine learning*. PMLR, 2016.

MC-Batch-normalization

- Batch normalization is used during training of DNNs to make the training stable and convergence faster.

- MC batch normalization selects a mini-batch B from training data D and performs batch normalization for each layer

- MC-Batchnorm also applies to testing by collecting the batch-normalization parameters from different mini-batches from training data to generate different outputs.
  - Sample different mini-batches from training data
  - Compute batch statistics (mean and variance) given sampled mini-batches
  - Run the same test input multiple times given batch statistics to get a distribution

Teye, Mattias, Hossein Azizpour, and Kevin Smith. "Bayesian uncertainty estimation for batch normalized deep networks." *International Conference on Machine Learning*. PMLR, 2018.

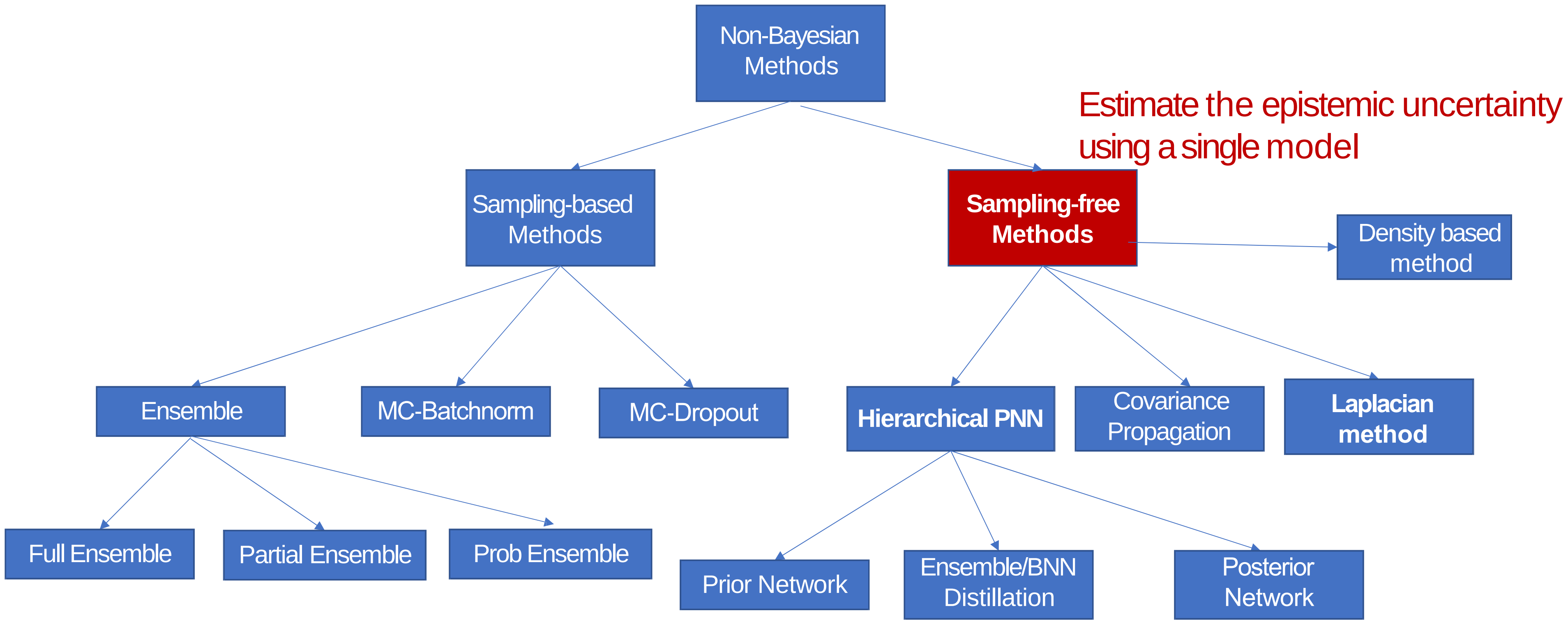
Sampling based methods comparison

- For ensemble methods:
  - Advantages: better uncertainty estimation than MC-dropout and MC-batchnorm
  - Disadvantages: larger computational and memory cost

- Dropout method
  - It is easy to turn an existing deep net into a Bayesian one., faster than other techniques
  - Sampling at test time might be too expensive for computationally-demanding (eg real time) applications.
  - it has worse uncertainty estimation than Ensemble methods, but the training cost is lower

- MC-Batchnorm:
  - More efficient than MC dropout
  - Accuracy is not good

Types of Non-Bayesian Uncertainty Estimation Methods



Non-Bayesian Methods

Sampling-based Methods

Sampling-free Methods

Estimate the epistemic uncertainty using a single model

Density based method

Ensemble

MC-Batchnorm

MC-Dropout

Hierarchical PNN

Covariance Propagation

Laplacian method

Full Ensemble

Partial Ensemble

Prob Ensemble

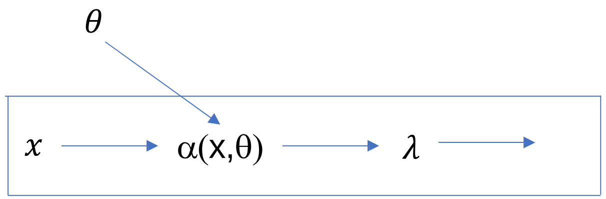
Prior Network

Ensemble/BNN Distillation

Posterior Network

Hierarchical Probabilistic Neural Network (HPNN)

HPNN Architecture

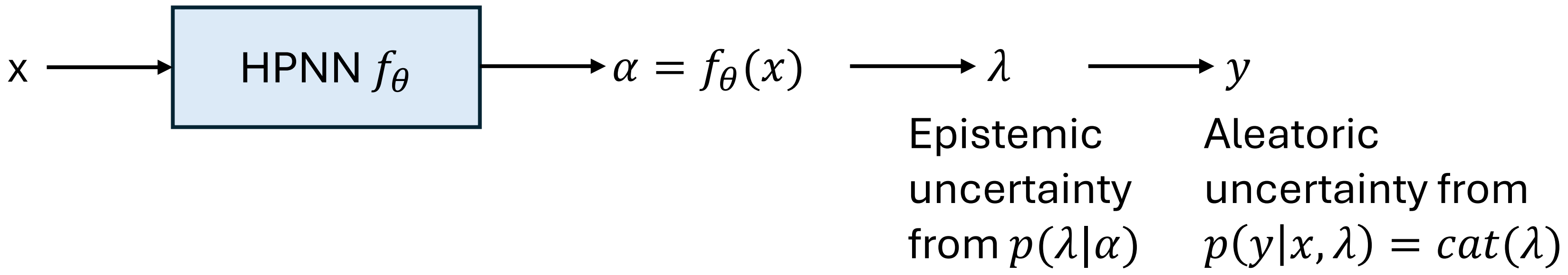


- θ is the parameter of the HPNN model
- $\alpha(x,\theta)$ are the output of HPNN and are the hyper-parameters that specify the prior distribution of λ, $p(\lambda|\alpha)$
- λ parameterizes the distribution of $p(y\,|\,x,\lambda)$ and are treated as random variables
- $p(y|x,\lambda)$ (the aleatoric distribution) can be used to estimate aleatoric uncertainty
- $p(\lambda|\alpha)$ (the epistemic distribution) can estimate the epistemic uncertainty

- It is a sampling-free single-network method and hence is efficient during inference.
- It requires a learning process to learn θ^* , the ML/MAP estimated parameters of HPNN.
- It can only model the parameter distribution around the estimated parameters θ^* (local uncertainty)

HPNN for Classification and Regression

For classification problem:



x

HPNN f_θ

$\alpha = f_\theta(x)$

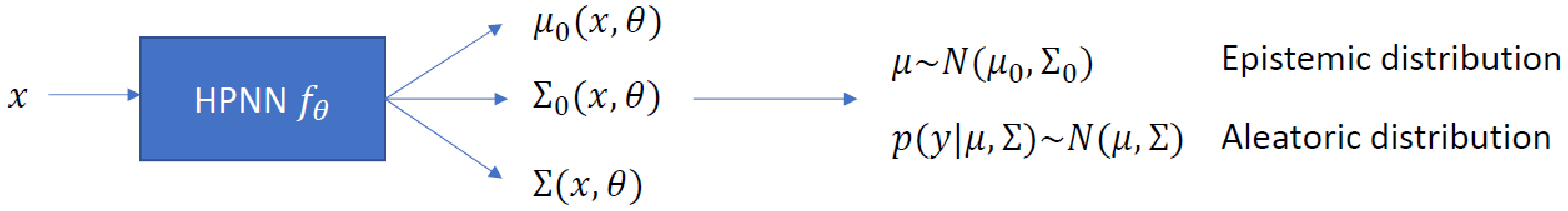
λ

y

Epistemic uncertainty from $p(\lambda|\alpha)$

Aleatoric uncertainty from $p(y|x,\lambda) = cat(\lambda)$

For regression problem:



x

HPNN f_θ

$\mu_0(x,\theta)$

$\Sigma_0(x,\theta)$

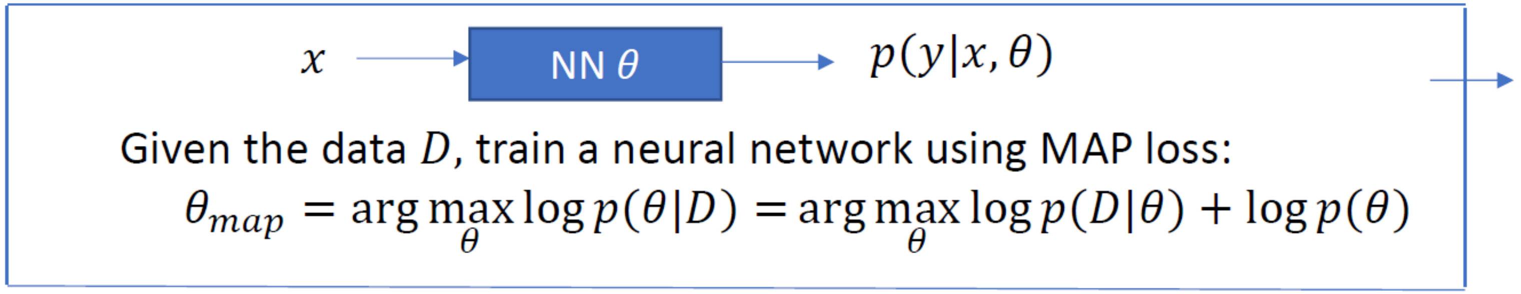
$\Sigma(x,\theta)$

$\mu \sim N(\mu_0, \Sigma_0)$ Epistemic distribution

$p(y|\mu,\Sigma) \sim N(\mu,\Sigma)$ Aleatoric distribution

Laplacian Approximation

- Can we train a single network to obtain different samples of the output distribution $p(y|x,\theta_1), p(y|x,\theta_1) \ldots, p(y|x,\theta_N)$? Yes, use Laplacian approximation



Given the data D, train a neural network using MAP loss:

$$\theta_{map} = \arg\max_{\theta} \log p(\theta|D) = \arg\max_{\theta} \log p(D|\theta) + \log p(\theta)$$

Pre-trained model

- Laplacian Approximation: a method to approximate $p(\theta|D)$ by a Gaussian distribution around the mode for the pretrained probabilistic neural networks.

By using the second-order Taylor expansion of $\log p(\theta|D)$ at θ_{map}

$$\log p(\theta|D) \approx \log p\left(\theta_{map}|D\right) + J(\theta - \theta_{map}) + \frac{1}{2}\left(\theta - \theta_{map}\right)^T H(\theta - \theta_{map})$$

$$J = \nabla_\theta \log p(\theta|D)\Big|_{\theta=\theta_{map}} = \mathbf{0} \qquad H = \nabla_\theta^2 \log p(\theta|D)\Big|_{\theta=\theta_{map}}$$

$$\Rightarrow \log p(\theta|D) \approx \log p\left(\theta_{map}|D\right) + \frac{1}{2}\left(\theta - \theta_{map}\right)^T H(\theta - \theta_{map})$$

$$\Rightarrow p(\theta|D) \approx p\left(\theta_{map}|D\right) \exp(\frac{1}{2}\left(\theta - \theta_{map}\right)^T H(\theta - \theta_{map}))$$

Laplacian Approximation – cont’d

$$\Longrightarrow p(\theta|D) \approx p\left(\theta_{map}|D\right) \underbrace{\exp(\frac{1}{2}(\theta - \theta_{map})^T H(\theta - \theta_{map}))}_{\text{Unnormalized multivariate Gaussian distribution}}$$

$$\Rightarrow \boxed{p(\theta|D) \approx \ N\left(\theta_{map}, (-H)^{-1}\right) ; H = \ \nabla_\theta^2 \log p(\theta|D)\Big|_{\theta=\theta_{map}}}$$

- During testing:

$$p(y^*|x^*, D) = \int p(y^*|x^*, \theta)p(\theta|D)d\theta \approx \frac{1}{S}\sum_{s=1}^{S} p(y^*|x^*, \theta_s) ; \ \ \theta_s \sim p(\theta|D) \approx N\left(\theta_{map}, (-H)^{-1}\right)$$

- For Laplacian approximation
  - It can be applied to any pre-trained single network. No retraining needed to generate uncertainty.
  - For large models, computing the inverse of Hessian matrix may be difficult. It may need some approximations. For example, only consider $diag(H)$.
  - It only approximates local parameter distribution around θ_{map}.

Bayesian Neural Network Software

The major sampling and variational methods have been implemented in the following software

- TensorFlow probability
  - https://www.tensorflow.org/probability

- Edward (A library for probabilistic modeling, inference, and criticism)
  - http://edwardlib.org/

- Pyro (Deep Universal Probabilistic Programming, supported by PyTorch)
  - https://pyro.ai/

- Gen (An open-source stack for generative modeling and probabilistic inference)
  - https://www.gen.dev/

Bayesian Neural Networks: Additional Materials

- A first insight into Bayesian Neural Network

https://medium.com/@costaleirbag/a-first-insight-into-bayesian-neural-networks-bnn-c767551e9526

- Introduction to full Bayesian approach

https://www.youtube.com/watch?v=jN5uYO9qllc

- The Bayesian interpretation of weight decay

https://www.youtube.com/watch?v=vEPQNwxd1Y4