CSCI 699 - ProbGen

Probabilistic and Generative Models

Willie Neiswanger

Lecture 11 - Predictive Uncertainty Quantification

Today

Today

Lecture: Predictive UQ models, including goals/setup/PGMs, Gaussian processes (GPs), and deep uncertainty models (BNNs, ensembles/dropout, etc.)

Today

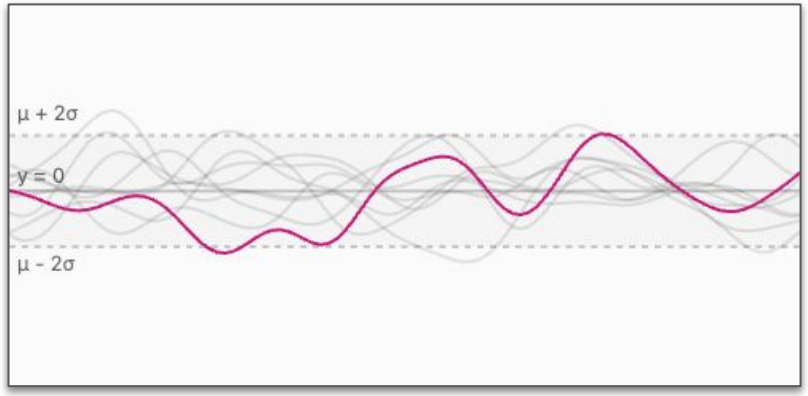
Lecture: Predictive UQ models, including goals/setup/PGMs, Gaussian processes (GPs), and deep uncertainty models (BNNs, ensembles/dropout, etc.)

- Motivation: Active learning, sequential experimental design, BO.

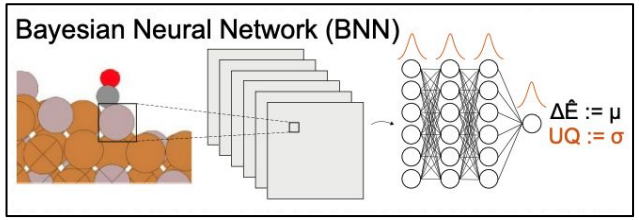
Today

Lecture: Predictive UQ models, including goals/setup/PGMs, Gaussian processes (GPs), and deep uncertainty models (BNNs, ensembles/dropout, etc.)

- Motivation: Active learning, sequential experimental design, BO.
- Classic predictive UQ: Bayesian parametric models, Gaussian processes (GPs).



A Visual Exploration of Gaussian Processes, distill.pub

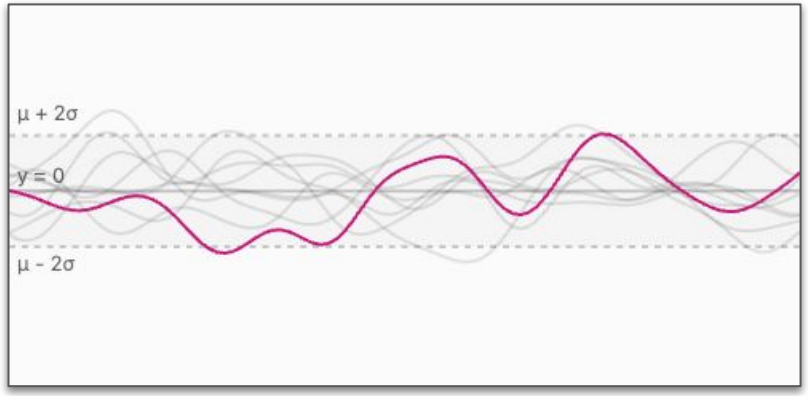


Tran, Neiswanger, et al., MLST. 2020

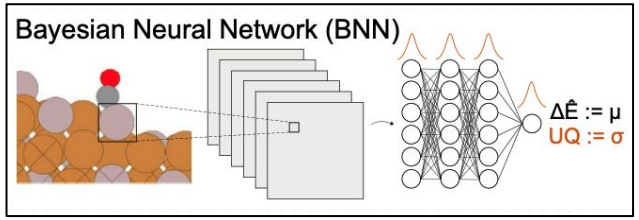
Today

Lecture: Predictive UQ models, including goals/setup/PGMs, Gaussian processes (GPs), and deep uncertainty models (BNNs, ensembles/dropout, etc.)

- Motivation: Active learning, sequential experimental design, BO.
- Classic predictive UQ: Bayesian parametric models, Gaussian processes (GPs).
- Neural/deep predictive UQ: Bayesian neural networks (BNN), ensembles, MC-dropout.



A Visual Exploration of Gaussian Processes, distill.pub

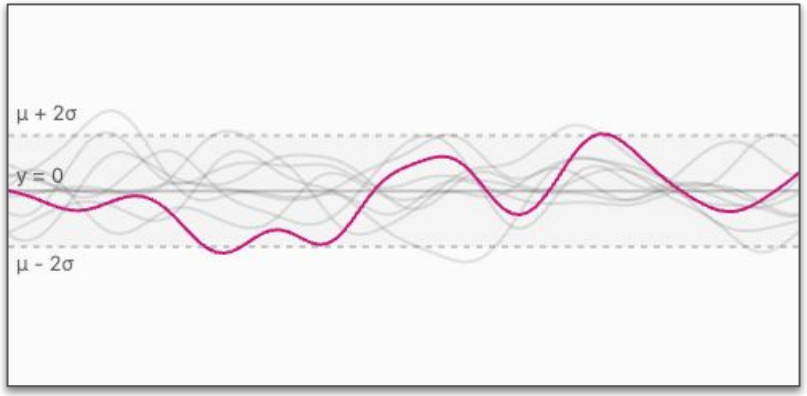


Tran, Neiswanger, et al., MLST. 2020

Today

Lecture: Predictive UQ models, including goals/setup/PGMs, Gaussian processes (GPs), and deep uncertainty models (BNNs, ensembles/dropout, etc.)

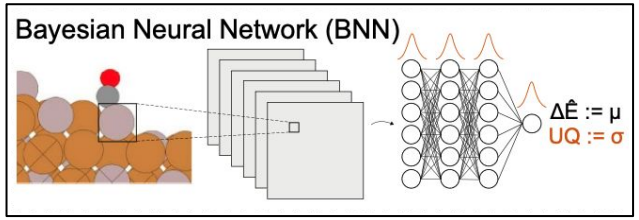
- Motivation: Active learning, sequential experimental design, BO.
- Classic predictive UQ: Bayesian parametric models, Gaussian processes (GPs).
- Neural/deep predictive UQ: Bayesian neural networks (BNN), ensembles, MC-dropout.



A Visual Exploration of Gaussian Processes, distill.pub

After:

- Next batch of four **paper presentations**.



Tran, Neiswanger, et al., MLST. 2020

Today

After: Fourth batch of four **paper presentations.**

Adding Conditional Control to Text-to-Image Diffusion Models

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala
Stanford University
{lvmin, anyirao, maneesh}@cs.stanford.edu

Input Canny edge | Default | "masterpiece of fairy tale, giant deer, golden antlers" | "..., quaint city Galic"

Input human pose | Default | "chef in kitchen" | "Lincoln statue"

Figure 1: Controlling Stable Diffusion with learned conditions. ControlNet allows users to add conditions like Canny edges (top), human pose (bottom), *etc.*, to control the image generation of large pretrained diffusion models. The default results use the prompt "a high-quality, detailed, and professional image". Users can optionally give prompts like the "chef in kitchen".

Abstract

We present ControlNet, a neural network architecture to add spatial conditioning controls to large, pretrained text-to-image diffusion models. ControlNet locks the production-ready large diffusion models, and reuses their deep and robust encoding layers pretrained with billions of images as a strong backbone to learn a diverse set of conditional controls. The neural architecture is connected with "zero convolutions" (zero-initialized convolution layers) that progressively grow the parameters from zero and ensure that no harmful noise could affect the finetuning. We test various conditioning controls, e.g., edges, depth, segmentation, human pose, etc., with Stable Diffusion, using single or multiple conditions, with or without prompts. We show that the training of ControlNets is robust with small (<50k) and large (>1m) datasets. Extensive results show that ControlNet may facilitate wider applications to control image diffusion models.

1. Introduction

Many of us have experienced flashes of visual inspiration that we wish to capture in a unique image. With the advent of text-to-image diffusion models [54, 62, 72], we can now create visually stunning images by typing in a text prompt. Yet, text-to-image models are limited in the control they provide over the spatial composition of the image: precisely expressing complex layouts, poses, shapes and forms can be difficult via text prompts alone. Generating an image that accurately matches our mental imagery often requires numerous trial-and-error cycles of editing a prompt, inspecting the resulting images and then re-editing the prompt.

Can we enable finer grained spatial control by letting users provide additional images that directly specify their desired image composition? In computer vision and machine learning, these additional images (*e.g.*, edge maps, human pose skeletons, segmentation maps, depth, normals, *etc.*) are often treated as conditioning on the image generation process. Image-to-image translation models [34, 98] learn

GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models

Jiaxuan You [*1] Rex Ying [*1] Xiang Ren [2] William L. Hamilton [1] Jure Leskovec [1]

Abstract

Modeling and generating graphs is fundamental for studying networks in biology, engineering, and social sciences. However, modeling complex distributions over graphs and then efficiently sampling from these distributions is challenging due to the non-unique, high-dimensional nature of graphs and the complex, non-local dependencies that exist between edges in a given graph. Here we propose GraphRNN, a deep autoregressive model that addresses the above challenges and approximates any distribution of graphs with minimal assumptions about their structure. GraphRNN learns to generate graphs by training on a representative set of graphs and decomposes the graph generation process into a sequence of node and edge formations, conditioned on the graph structure generated so far. In order to quantitatively evaluate the performance of GraphRNN, we introduce a benchmark suite of datasets, baselines and novel evaluation metrics based on Maximum Mean Discrepancy, which measure distances between sets of graphs. Our experiments show that GraphRNN significantly outperforms all baselines, learning to generate diverse graphs that match the structural characteristics of a target set, while also scaling to graphs $50\times$ larger than previous deep models.

1. Introduction and Related Work

Generative models for real-world graphs have important applications in many domains, including modeling physical and social interactions, discovering new chemical and molecular structures, and constructing knowledge graphs. Development of generative graph models has a rich history, and many methods have been proposed that can generate graphs based on a priori structural assumptions (Newman, 2010). However, a key open challenge in this area is developing methods that can directly *learn* generative models from an observed set of graphs. Developing generative models that can learn directly from data is an important step towards improving the fidelity of generated graphs, and paves a way for new kinds of applications, such as discovering new graph structures and completing evolving graphs.

In contrast, traditional generative models for graphs (*e.g.*, Barabási-Albert model, Kronecker graphs, exponential random graphs, and stochastic block models) (Erdős & Rényi, 1959; Leskovec et al., 2010; Albert & Barabási, 2002; Airoldi et al., 2008; Leskovec et al., 2007; Robins et al., 2007) are hand-engineered to model a particular family of graphs, and thus do not have the capacity to directly learn the generative model from observed data. For example, the Barabási-Albert model is carefully designed to capture the scale-free nature of empirical degree distributions, but fails to capture many other aspects of real-world graphs, such as community structure..

Recent advances in deep generative models, such as variational autoencoders (VAE) (Kingma & Welling, 2014) and generative adversarial networks (GAN) (Goodfellow et al., 2014), have made important progress towards generative modeling for complex domains, such as image and text data. Building on these approaches a number of deep learning models for generating graphs have been proposed (Kipf & Welling, 2016; Grover et al., 2017; Simonovsky & Komodakis, 2018; Li et al., 2018). For example, Simonovsky & Komodakis 2018 propose a VAE-based approach, while Li et al. 2018 propose a framework based upon graph neural networks. However, these recently proposed deep models are either limited to learning from a single graph (Kipf & Welling, 2016; Grover et al., 2017) or generating small graphs with 40 or fewer nodes (Li et al., 2018; Simonovsky & Komodakis, 2018)—limitations that stem from three fundamental challenges in the graph generation problem:

- **Large and variable output spaces:** To generate a graph with n nodes the generative model has to output n^2 values to fully specify its structure. Also, the number of nodes n and edges m varies between different graphs and a generative model needs to accommodate such complexity and variability in the output space.

[*]Equal contribution [1]Department of Computer Science, Stanford University, Stanford, CA, 94305 [2]Department of Computer Science, University of Southern California, Los Angeles, CA, 90007. Correspondence to: Jiaxuan You <jiaxuan@stanford.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

Published as a conference paper at ICLR 2025

JUDGE DECODING: FASTER SPECULATIVE SAMPLING REQUIRES GOING BEYOND MODEL ALIGNMENT

Gregor Bachmann[†,♯,*] Sotiris Anagnostidis[†,♯] Albert Pumarola[†]
Markos Georgopoulos[†] Artsiom Sanakoyeu[†] Yuming Du[†]
Edgar Schönfeld[†] Ali Thabet[†] Jonas Kohler[†]

ABSTRACT

The performance of large language models (LLMs) is closely linked to their underlying size, leading to ever-growing networks and hence slower inference. Speculative decoding has been proposed as a technique to accelerate autoregressive generation, leveraging a fast draft model to propose candidate tokens, which are then verified in parallel based on their likelihood under the target model. While this approach guarantees to reproduce the target output, it incurs a substantial penalty: many high-quality draft tokens are rejected, even when they represent objectively valid continuations. Indeed, we show that even powerful draft models such as `GPT-4o`, as well as human text cannot achieve high acceptance rates under the standard verification scheme. This severely limits the speedup potential of current speculative decoding methods, as an early rejection becomes overwhelmingly likely when solely relying on alignment of draft and target.
We thus ask the following question: *Can we adapt verification to recognize correct, but non-aligned replies?* To this end, we draw inspiration from the *LLM-as-a-judge* framework, which demonstrated that LLMs are able to rate answers in a versatile way. We carefully design a dataset to elicit the same capability in the target model by training a compact module on top of the embeddings to produce "judgements" of the current continuation. We showcase our strategy on the `Llama-3.1` family, where our `8b/405B-Judge` achieves a speedup of $9\times$ over `Llama-405B`, while maintaining its quality on a large range of benchmarks. These benefits remain present even in optimized inference frameworks, where our method reaches up to 141 tokens/s for `8B/70B-Judge` and 129 tokens/s for `8B/405B` on 2 and 8 H100s respectively.

Figure 1: Standard speculative decoding versus our judge decoding strategy for `Llama-3.1-8B` as draft and `Llama-3.1-405B` as target. Accepted (rejected) tokens are highlighted in green (red).

[*]Work done during an internship at Meta GenAI. [†]Meta GenAI. [♯]ETH Zürich.
Correspondence to: `gregorb@ethz.ch`

LoRA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

Edward Hu* Yelong Shen* Phillip Wallis Zeyuan Allen-Zhu
Yuanzhi Li Shean Wang Lu Wang Weizhu Chen
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

ABSTRACT

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Low-Rank Adaptation**, or **LoRA**, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at `https://github.com/microsoft/LoRA`.

1 INTRODUCTION

Many applications in natural language processing rely on adapting *one* large-scale, pre-trained language model to *multiple* downstream applications. Such adaptation is usually done via *fine-tuning*, which updates all the parameters of the pre-trained model. The major downside of fine-tuning is that the new model contains as many parameters as in the original model. As larger models are trained every few months, this changes from a mere "inconvenience" for GPT-2 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a critical deployment challenge for GPT-3 (Brown et al., 2020) with 175 billion trainable parameters.[1]

Many sought to mitigate this by adapting only some parameters or learning external modules for new tasks. This way, we only need to store and load a small number of task-specific parameters in addition to the pre-trained model for each task, greatly boosting the operational efficiency when deployed. However, existing techniques

Figure 1: Our reparametrization. We only train A and B.

[*]Equal contribution.
[3]Compared to V1, this draft includes better baselines, experiments on GLUE, and more on adapter latency.
[1]While GPT-3 175B achieves non-trivial performance with few-shot learning, fine-tuning boosts its performance significantly as shown in Appendix A.

Today

After: Fourth batch of four **paper presentations**.

Adding Conditional Control to Text-to-Image Diffusion Models

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala
Stanford University
{lvmin, anyirao, maneesh}@cs.stanford.edu

Input Canny edge | Default | "masterpiece of fairy tale, giant deer, golden antlers" | "..., quaint city Galic"

Input human pose | Default | "chef in kitchen" | "Lincoln statue"

Figure 1: Controlling Stable Diffusion with learned conditions. ControlNet allows users to add conditions like Canny edges (top), human pose (bottom), *etc.*, to control the image generation of large pretrained diffusion models. The default results use the prompt "a high-quality, detailed, and professional image". Users can optionally give prompts like the "chef in kitchen".

Abstract

We present ControlNet, a neural network architecture to add spatial conditioning controls to large, pretrained text-to-image diffusion models. ControlNet locks the production-ready large diffusion models, and reuses their deep and robust encoding layers pretrained with billions of images as a strong backbone to learn a diverse set of conditional controls. The neural architecture is connected with "zero convolutions" (zero-initialized convolution layers) that progressively grow the parameters from zero and ensure that no harmful noise could affect the finetuning. We test various conditioning controls, e.g., edges, depth, segmentation, human pose, etc., with Stable Diffusion, using single or multiple conditions, with or without prompts. We show that the training of ControlNets is robust with small (<50k) and large (>1m) datasets. Extensive results show that ControlNet may facilitate wider applications to control image diffusion models.

1. Introduction

Many of us have experienced flashes of visual inspiration that we wish to capture in a unique image. With the advent of text-to-image diffusion models [54, 62, 72], we can now create visually stunning images by typing in a text prompt. Yet, text-to-image models are limited in the control they provide over the spatial composition of the image: precisely expressing complex layouts, poses, shapes and forms can be difficult via text prompts alone. Generating an image that accurately matches our mental imagery often requires numerous trial-and-error cycles of editing a prompt, inspecting the resulting images and then re-editing the prompt.

Can we enable finer grained spatial control by letting users provide additional images that directly specify their desired image composition? In computer vision and machine learning, these additional images (*e.g.*, edge maps, human pose skeletons, segmentation maps, depth, normals, *etc.*) are often treated as conditioning on the image generation process. Image-to-image translation models [34, 98] learn

GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models

Jiaxuan You [*1] Rex Ying [*1] Xiang Ren [2] William L. Hamilton [1] Jure Leskovec [1]

Abstract

Modeling and generating graphs is fundamental for studying networks in biology, engineering, and social sciences. However, modeling complex distributions over graphs and then efficiently sampling from these distributions is challenging due to the non-unique, high-dimensional nature of graphs and the complex, non-local dependencies that exist between edges in a given graph. Here we propose GraphRNN, a deep autoregressive model that addresses the above challenges and approximates any distribution of graphs with minimal assumptions about their structure. GraphRNN learns to generate graphs by training on a representative set of graphs and decomposes the graph generation process into a sequence of node and edge formations, conditioned on the graph structure generated so far. In order to quantitatively evaluate the performance of GraphRNN, we introduce a benchmark suite of datasets, baselines and novel evaluation metrics based on Maximum Mean Discrepancy, which measure distances between sets of graphs. Our experiments show that GraphRNN significantly outperforms all baselines, learning to generate diverse graphs that match the structural characteristics of a target set, while also scaling to graphs $50\times$ larger than previous deep models.

1. Introduction and Related Work

Generative models for real-world graphs have important applications in many domains, including modeling physical and social interactions, discovering new chemical and molecular structures, and constructing knowledge graphs. Development of generative graph models has a rich history, and many methods have been proposed that can generate graphs based on a priori structural assumptions (Newman, 2010). However, a key open challenge in this area is developing methods that can directly *learn* generative models from an observed set of graphs. Developing generative models that can learn directly from data is an important step towards improving the fidelity of generated graphs, and paves a way for new kinds of applications, such as discovering new graph structures and completing evolving graphs.

In contrast, traditional generative models for graphs (*e.g.*, Barabási-Albert model, Kronecker graphs, exponential random graphs, and stochastic block models) (Erdős & Rényi, 1959; Leskovec et al., 2010; Albert & Barabási, 2002; Airoldi et al., 2008; Leskovec et al., 2007; Robins et al., 2007) are hand-engineered to model a particular family of graphs, and thus do not have the capacity to directly learn the generative model from observed data. For example, the Barabási-Albert model is carefully designed to capture the scale-free nature of empirical degree distributions, but fails to capture many other aspects of real-world graphs, such as community structure.

Recent advances in deep generative models, such as variational autoencoders (VAE) (Kingma & Welling, 2014) and generative adversarial networks (GAN) (Goodfellow et al., 2014), have made important progress towards generative modeling for complex domains, such as image and text data. Building on these approaches a number of deep learning models for generating graphs have been proposed (Kipf & Welling, 2016; Grover et al., 2017; Simonovsky & Komodakis, 2018; Li et al., 2018). For example, Simonovsky & Komodakis 2018 propose a VAE-based approach, while Li et al. 2018 propose a framework based upon graph neural networks. However, these recently proposed deep models are either limited to learning from a single graph (Kipf & Welling, 2016; Grover et al., 2017) or generating small graphs with 40 or fewer nodes (Li et al., 2018; Simonovsky & Komodakis, 2018)—limitations that stem from three fundamental challenges in the graph generation problem:

- **Large and variable output spaces:** To generate a graph with n nodes the generative model has to output n^2 values to fully specify its structure. Also, the number of nodes n and edges m varies between different graphs and a generative model needs to accommodate such complexity and variability in the output space.

[*]Equal contribution [1]Department of Computer Science, Stanford University, Stanford, CA, 94305 [2]Department of Computer Science, University of Southern California, Los Angeles, CA, 90007. Correspondence to: Jiaxuan You <jiaxuan@stanford.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

JUDGE DECODING: FASTER SPECULATIVE SAMPLING REQUIRES GOING BEYOND MODEL ALIGNMENT

Gregor Bachmann[†,♣,*] Sotiris Anagnostidis[†,♣] Albert Pumarola[†]
Markos Georgopoulos[†] Artsiom Sanakoyeu[†] Yuming Du[†]
Edgar Schönfeld[†] Ali Thabet[†] Jonas Kohler[†]

ABSTRACT

The performance of large language models (LLMs) is closely linked to their underlying size, leading to ever-growing networks and hence slower inference. Speculative decoding has been proposed as a technique to accelerate autoregressive generation, leveraging a fast draft model to propose candidate tokens, which are then verified in parallel based on their likelihood under the target model. While this approach guarantees to reproduce the target output, it incurs a substantial penalty: many high-quality draft tokens are rejected, even when they represent objectively valid continuations. Indeed, we show that even powerful draft models such as `GPT-4o`, as well as human text cannot achieve high acceptance rates under the standard verification scheme. This severely limits the speedup potential of current speculative decoding methods, as an early rejection becomes overwhelmingly likely when solely relying on alignment of draft and target.
We thus ask the following question: *Can we adapt verification to recognize correct, but non-aligned replies?* To this end, we draw inspiration from the *LLM-as-a-judge* framework, which demonstrated that LLMs are able to rate answers in a versatile way. We carefully design a dataset to elicit the same capability in the target model by training a compact module on top of the embeddings to produce "judgements" of the current continuation. We showcase our strategy on the `Llama-3.1` family, where our `8b/405B-Judge` achieves a speedup of $9\times$ over `Llama-405B`, while maintaining its quality on a large range of benchmarks. These benefits remain present even in optimized inference frameworks, where our method reaches up to 141 tokens/s for `8B/70B-Judge` and 129 tokens/s for `8B/405B` on 2 and 8 H100s respectively.

Figure 1: Standard speculative decoding versus our judge decoding strategy for `Llama-3.1-8B` as draft and `Llama-3.1-405B` as target. Accepted (rejected) tokens are highlighted in green (red).

[*]Work done during an internship at Meta GenAI. [♣]Meta GenAI. [†]ETH Zürich.
Correspondence to: gregorb@ethz.ch

LoRA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

Edward Hu* Yelong Shen* Phillip Wallis Zeyuan Allen-Zhu
Yuanzhi Li Shean Wang Lu Wang Weizhu Chen
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

ABSTRACT

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Low-Rank Adaptation**, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at `https://github.com/microsoft/LoRA`.

1 INTRODUCTION

Many applications in natural language processing rely on adapting *one* large-scale, pre-trained language model to *multiple* downstream applications. Such adaptation is usually done via *fine-tuning*, which updates all the parameters of the pre-trained model. The major downside of fine-tuning is that the new model contains as many parameters as in the original model. As larger models are trained every few months, this changes from a mere "inconvenience" for GPT-2 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a critical deployment challenge for GPT-3 (Brown et al., 2020) with 175 billion trainable parameters.[1]

Many sought to mitigate this by adapting only some parameters or learning external modules for new tasks. This way, we only need to store and load a small number of task-specific parameters in addition to the pre-trained model for each task, greatly boosting the operational efficiency when deployed. However, existing techniques

Figure 1: Our reparametrization. We only train A and B.

[*]Equal contribution.
[3]Compared to V1, this draft includes better baselines, experiments on GLUE, and more on adapter latency.
[1]While GPT-3 175B achieves non-trivial performance with few-shot learning, fine-tuning boosts its performance significantly as shown in Appendix A.

ControlNet — an important conditional gen paper

Today

After: Fourth batch of four **paper presentations**.

arXiv:2302.05543v3 [cs.CV] 26 Nov 2023

Adding Conditional Control to Text-to-Image Diffusion Models

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala
Stanford University
{lvmin, anyirao, maneesh}@cs.stanford.edu

Input Canny edge | Default | "masterpiece of fairy tale, giant deer, golden antlers" | "... quaint city Galic"

Input human pose | Default | "chef in kitchen" | "Lincoln statue"

Figure 1: Controlling Stable Diffusion with learned conditions. ControlNet allows users to add conditions like Canny edges (top), human pose (bottom), *etc.*, to control the image generation of large pretrained diffusion models. The default results use the prompt "a high-quality, detailed, and professional image". Users can optionally give prompts like the "chef in kitchen".

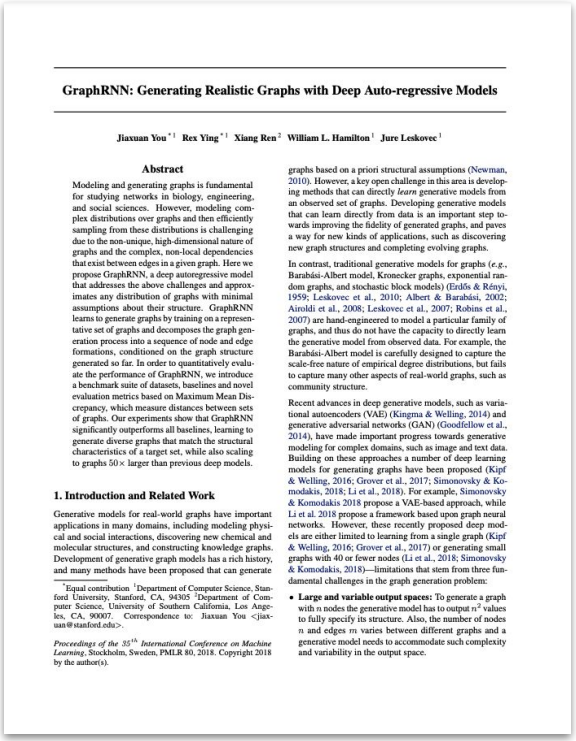
Abstract

We present ControlNet, a neural network architecture to add spatial conditioning controls to large, pretrained text-to-image diffusion models. ControlNet locks the production-ready large diffusion models, and reuses their deep and robust encoding layers pretrained with billions of images as a strong backbone to learn a diverse set of conditional controls. The neural architecture is connected with "zero convolutions" (zero-initialized convolution layers) that progressively grow the parameters from zero and ensure that no harmful noise could affect the finetuning. We test various conditioning controls, e.g., edges, depth, segmentation, human pose, etc., with Stable Diffusion, using single or multiple conditions, with or without prompts. We show that the training of ControlNets is robust with small (<50k) and large (>1m) datasets. Extensive results show that ControlNet may facilitate wider applications to control image diffusion models.

1. Introduction

Many of us have experienced flashes of visual inspiration that we wish to capture in a unique image. With the advent of text-to-image diffusion models [54, 62, 72], we can now create visually stunning images by typing in a text prompt. Yet, text-to-image models are limited in the control they provide over the spatial composition of the image: precisely expressing complex layouts, poses, shapes and forms can be difficult via text prompts alone. Generating an image that accurately matches our mental imagery often requires numerous trial-and-error cycles of editing a prompt, inspecting the resulting images and then re-editing the prompt.

Can we enable finer grained spatial control by letting users provide additional images that directly specify their desired image composition? In computer vision and machine learning, these additional images (*e.g.*, edge maps, human pose skeletons, segmentation maps, depth, normals, *etc.*) are often treated as conditioning on the image generation process. Image-to-image translation models [34, 98] learn



GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models

Jiaxuan You [*1] Rex Ying [*1] Xiang Ren [2] William L. Hamilton [1] Jure Leskovec [1]

Abstract

Modeling and generating graphs is fundamental for studying networks in biology, engineering, and social sciences. However, modeling complex distributions over graphs and then efficiently sampling from these distributions is challenging due to the non-unique, high-dimensional nature of graphs and the complex, non-local dependencies that exist between edges in a given graph. Here we propose GraphRNN, a deep autoregressive model that addresses the above challenges and approximates any distribution of graphs with minimal assumptions about their structure. GraphRNN learns to generate graphs by training on a representative set of graphs and decomposes the graph generation process into a sequence of node and edge formations, conditioned on the graph structure generated so far. In order to quantitatively evaluate the performance of GraphRNN, we introduce a benchmark suite of datasets, baselines and novel evaluation metrics based on Maximum Mean Discrepancy, which measure distances between sets of graphs. Our experiments show that GraphRNN significantly outperforms all baselines, learning to generate diverse graphs that match the structural characteristics of a target set, while also scaling to graphs $50\times$ larger than previous deep models.

1. Introduction and Related Work

Generative models for real-world graphs have important applications in many domains, including modeling physical and social interactions, discovering new chemical and molecular structures, and constructing knowledge graphs. Development of generative graph models has a rich history, and many methods have been proposed that can generate graphs based on a priori structural assumptions (Newman, 2010). However, a key open challenge in this area is developing methods that can directly *learn* generative models from an observed set of graphs. Developing generative models that can learn directly from data is an important step towards improving the fidelity of generated graphs, and paves a way for new kinds of applications, such as discovering new graph structures and completing evolving graphs.

In contrast, traditional generative models for graphs (*e.g.*, Barabási-Albert model, Kronecker graphs, exponential random graphs, and stochastic block models) (Erdős & Rényi, 1959; Leskovec et al., 2010; Albert & Barabási, 2002; Airoldi et al., 2008; Leskovec et al., 2007; Robins et al., 2007) are hand-engineered to model a particular family of graphs, and thus do not have the capacity to directly learn the generative model from observed data. For example, the Barabási-Albert model is carefully designed to capture the scale-free nature of empirical degree distributions, but fails to capture many other aspects of real-world graphs, such as community structure.

Recent advances in deep generative models, such as variational autoencoders (VAE) (Kingma & Welling, 2014) and generative adversarial networks (GAN) (Goodfellow et al., 2014), have made important progress towards generative modeling for complex domains, such as image and text data. Building on these approaches a number of deep learning models for generating graphs have been proposed (Kipf & Welling, 2016; Grover et al., 2017; Simonovsky & Komodakis, 2018; Li et al., 2018). For example, Simonovsky & Komodakis 2018 propose a VAE-based approach, while Li et al. 2018 propose a framework based upon graph neural networks. However, these recently proposed deep models are either limited to learning from a single graph (Kipf & Welling, 2016; Grover et al., 2017) or generating small graphs with 40 or fewer nodes (Li et al., 2018; Simonovsky & Komodakis, 2018)—limitations that stem from two fundamental challenges in the graph generation problem:

- **Large and variable output spaces:** To generate a graph with n nodes the generative model has to output n^2 values to fully specify its structure. Also, the number of nodes n and edges m varies between different graphs and a generative model needs to accommodate such complexity and variability in the output space.

[*]Equal contribution [1]Department of Computer Science, Stanford University, Stanford, CA, 94305 [2]Department of Computer Science, University of Southern California, Los Angeles, CA, 90007. Correspondence to: Jiaxuan You <jiaxuan@stanford.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

arXiv:2501.19309v1 [cs.LG] 31 Jan 2025

Published as a conference paper at ICLR 2025

JUDGE DECODING: FASTER SPECULATIVE SAMPLING REQUIRES GOING BEYOND MODEL ALIGNMENT

Gregor Bachmann[†,♣,*] Sotiris Anagnostidis[†,♣] Albert Pumarola[†]
Markos Georgopoulos[†] Artsiom Sanakoyeu[†] Yuming Du[†]
Edgar Schönfeld[†] Ali Thabet[†] Jonas Kohler[†]

ABSTRACT

The performance of large language models (LLMs) is closely linked to their underlying size, leading to ever-growing networks and hence slower inference. Speculative decoding has been proposed as a technique to accelerate autoregressive generation, leveraging a fast draft model to propose candidate tokens, which are then verified in parallel based on their likelihood under the target model. While this approach guarantees to reproduce the target output, it incurs a substantial penalty: many high-quality draft tokens are rejected, even when they represent objectively valid continuations. Indeed, we show that even powerful draft models such as `GPT-4o`, as well as human text cannot achieve high acceptance rates under the standard verification scheme. This severely limits the speedup potential of current speculative decoding methods, as an early rejection becomes overwhelmingly likely when solely relying on alignment of draft and target.
We thus ask the following question: *Can we adapt verification to recognize correct, but non-aligned replies?* To this end, we draw inspiration from the *LLM-as-a-judge* framework, which demonstrated that LLMs are able to rate answers in a versatile way. We carefully design a dataset to elicit the same capability in the target model by training a compact module on top of the embeddings to produce "judgements" of the current continuation. We showcase our strategy on the `Llama-3.1` family, where our `8b/405B-Judge` achieves a speedup of $9\times$ over `Llama-405B`, while maintaining its quality on a large range of benchmarks. These benefits remain present even in optimized inference frameworks, where our method reaches up to 141 tokens/s for `8B/70B-Judge` and 129 tokens/s for `8B/405B` on 2 and 8 H100s respectively.

Figure 1: Standard speculative decoding versus our judge decoding strategy for `Llama-3.1-8B` as draft and `Llama-3.1-405B` as target. Accepted (rejected) tokens are highlighted in green (red).

[*]Work done during an internship at Meta GenAI. [♣]Meta GenAI. [†]ETH Zürich.
Correspondence to: `gregorb@ethz.ch`

arXiv:2106.09685v2 [cs.CL] 16 Oct 2021

LoRA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

Edward Hu* Yelong Shen* Phillip Wallis Zeyuan Allen-Zhu
Yuanzhi Li Shean Wang Lu Wang Weizhu Chen
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

ABSTRACT

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Low-Rank** Adaptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at `https://github.com/microsoft/LoRA`.

1 INTRODUCTION

Many applications in natural language processing rely on adapting *one* large-scale, pre-trained language model to *multiple* downstream applications. Such adaptation is usually done via *fine-tuning*, which updates all the parameters of the pre-trained model. The major downside of fine-tuning is that the new model contains as many parameters as in the original model. As larger models are trained every few months, this changes from a mere "inconvenience" for GPT-2 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a critical deployment challenge for GPT-3 (Brown et al., 2020) with 175 billion trainable parameters.[1]

Many sought to mitigate this by adapting only some parameters or learning external modules for new tasks. This way, we only need to store and load a small number of task-specific parameters in addition to the pre-trained model for each task, greatly boosting the operational efficiency when deployed. However, existing techniques

Figure 1: Our reparametrization. We only train A and B.

*Equal contribution.
[†]Compared to V1, this draft includes better baselines, experiments on GLUE, and more on adapter latency.
[1]While GPT-3 175B achieves non-trivial performance with few-shot learning, fine-tuning boosts its performance significantly as shown in Appendix A.

GraphRNN: generate graphs via autoregression

Today

After: Fourth batch of four **paper presentations**.

Adding Conditional Control to Text-to-Image Diffusion Models

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala
Stanford University
{lvmin, anyirao, maneesh}@cs.stanford.edu

Input Canny edge | Default | "masterpiece of fairy tale, giant deer, golden antlers" | "..., quaint city Galic"

Input human pose | Default | "chef in kitchen" | "Lincoln statue"

Figure 1: Controlling Stable Diffusion with learned conditions. ControlNet allows users to add conditions like Canny edges (top), human pose (bottom), *etc.*, to control the image generation of large pretrained diffusion models. The default results use the prompt "a high-quality, detailed, and professional image". Users can optionally give prompts like the "chef in kitchen".

Abstract

We present ControlNet, a neural network architecture to add spatial conditioning controls to large, pretrained text-to-image diffusion models. ControlNet locks the production-ready large diffusion models, and reuses their deep and robust encoding layers pretrained with billions of images as a strong backbone to learn a diverse set of conditional controls. The neural architecture is connected with "zero convolutions" (zero-initialized convolution layers) that progressively grow the parameters from zero and ensure that no harmful noise could affect the finetuning. We test various conditioning controls, e.g., edges, depth, segmentation, human pose, etc., with Stable Diffusion, using single or multiple conditions, with or without prompts. We show that the training of ControlNets is robust with small (<50k) and large (>1m) datasets. Extensive results show that ControlNet may facilitate wider applications to control image diffusion models.

1. Introduction

Many of us have experienced flashes of visual inspiration that we wish to capture in a unique image. With the advent of text-to-image diffusion models [54, 62, 72], we can now create visually stunning images by typing in a text prompt. Yet, text-to-image models are limited in the control they provide over the spatial composition of the image: precisely expressing complex layouts, poses, shapes and forms can be difficult via text prompts alone. Generating an image that accurately matches our mental imagery often requires numerous trial-and-error cycles of editing a prompt, inspecting the resulting images and then re-editing the prompt.

Can we enable finer grained spatial control by letting users provide additional images that directly specify their desired image composition? In computer vision and machine learning, these additional images (*e.g.*, edge maps, human pose skeletons, segmentation maps, depth, normals, *etc.*) are often treated as conditioning on the image generation process. Image-to-image translation models [34, 98] learn

GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models

Jiaxuan You [*1] Rex Ying [*1] Xiang Ren [2] William L. Hamilton [1] Jure Leskovec [1]

Abstract

Modeling and generating graphs is fundamental for studying networks in biology, engineering, and social sciences. However, modeling complex distributions over graphs and then efficiently sampling from these distributions is challenging due to the non-unique, high-dimensional nature of graphs and the complex, non-local dependencies that exist between edges in a given graph. Here we propose GraphRNN, a deep autoregressive model that addresses the above challenges and approximates any distribution of graphs with minimal assumptions about their structure. GraphRNN learns to generate graphs by training on a representative set of graphs and decomposes the graph generation process into a sequence of node and edge formations, conditioned on the graph structure generated so far. In order to quantitatively evaluate the performance of GraphRNN, we introduce a benchmark suite of datasets, baselines and novel evaluation metrics based on Maximum Mean Discrepancy, which measure distances between sets of graphs. Our experiments show that GraphRNN significantly outperforms all baselines, learning to generate diverse graphs that match the structural characteristics of a target set, while also scaling to graphs $50\times$ larger than previous deep models.

1. Introduction and Related Work

Generative models for real-world graphs have important applications in many domains, including modeling physical and social interactions, discovering new chemical and molecular structures, and constructing knowledge graphs. Development of generative graph models has a rich history, and many methods have been proposed that can generate graphs based on a priori structural assumptions (Newman, 2010). However, a key open challenge in this area is developing methods that can directly *learn* generative models from an observed set of graphs. Developing generative models that can learn directly from data is an important step towards improving the fidelity of generated graphs, and paves a way for new kinds of applications, such as discovering new graph structures and completing evolving graphs.

In contrast, traditional generative models for graphs (*e.g.*, Barabási-Albert model, Kronecker graphs, exponential random graphs, and stochastic block models) (Erdős & Rényi, 1959; Leskovec et al., 2010; Albert & Barabási, 2002; Airoldi et al., 2008; Leskovec et al., 2007; Robins et al., 2007) are hand-engineered to model a particular family of graphs, and thus do not have the capacity to directly learn the generative model from observed data. For example, the Barabási-Albert model is carefully designed to capture the scale-free nature of empirical degree distributions, but fails to capture many other aspects of real-world graphs, such as community structure.

Recent advances in deep generative models, such as variational autoencoders (VAE) (Kingma & Welling, 2014) and generative adversarial networks (GAN) (Goodfellow et al., 2014), have made important progress towards generative modeling for complex domains, such as image and text data. Building on these approaches a number of deep learning models for generating graphs have been proposed (Kipf & Welling, 2016; Grover et al., 2017; Simonovsky & Komodakis, 2018; Li et al., 2018). For example, Simonovsky & Komodakis 2018 propose a VAE-based approach, while Li et al. 2018 propose a framework based upon graph neural networks. However, these recently proposed deep models are either limited to learning from a single graph (Kipf & Welling, 2016; Grover et al., 2017) or generating small graphs with 40 or fewer nodes (Li et al., 2018; Simonovsky & Komodakis, 2018)—limitations that stem from three fundamental challenges in the graph generation problem:

- **Large and variable output spaces:** To generate a graph with n nodes the generative model has to output n^2 values to fully specify its structure. Also, the number of nodes n and edges m varies between different graphs and a generative model needs to accommodate such complexity and variability in the output space.

[*]Equal contribution [1]Department of Computer Science, Stanford University, Stanford, CA, 94305 [2]Department of Computer Science, University of Southern California, Los Angeles, CA, 90007. Correspondence to: Jiaxuan You <jiaxuan@stanford.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

Published as a conference paper at ICLR 2025

JUDGE DECODING: FASTER SPECULATIVE SAMPLING REQUIRES GOING BEYOND MODEL ALIGNMENT

Gregor Bachmann[†,♣,*] Sotiris Anagnostidis[†,♣] Albert Pumarola[†]
Markos Georgopoulos[†] Artsiom Sanakoyeu[†] Yuming Du[†]
Edgar Schönfeld[†] Ali Thabet[†] Jonas Kohler[†]

ABSTRACT

The performance of large language models (LLMs) is closely linked to their underlying size, leading to ever-growing networks and hence slower inference. Speculative decoding has been proposed as a technique to accelerate autoregressive generation, leveraging a fast draft model to propose candidate tokens, which are then verified in parallel based on their likelihood under the target model. While this approach guarantees to reproduce the target output, it incurs a substantial penalty: many high-quality draft tokens are rejected, even when they represent objectively valid continuations. Indeed, we show that even powerful draft models such as `GPT-4o`, as well as human text cannot achieve high acceptance rates under the standard verification scheme. This severely limits the speedup potential of current speculative decoding methods, as an early rejection becomes overwhelmingly likely when solely relying on alignment of draft and target.
We thus ask the following question: *Can we adapt verification to recognize correct, but non-aligned replies?* To this end, we draw inspiration from the *LLM-as-a-judge* framework, which demonstrated that LLMs are able to rate answers in a versatile way. We carefully design a dataset to elicit the same capability in the target model by training a compact module on top of the embeddings to produce "judgements" of the current continuation. We showcase our strategy on the `Llama-3.1` family, where our `8b/405B-Judge` achieves a speedup of $9\times$ over `Llama-405B`, while maintaining its quality on a large range of benchmarks. These benefits remain present even in optimized inference frameworks, where our method reaches up to 141 tokens/s for `8B/70B-Judge` and 129 tokens/s for `8B/405B` on 2 and 8 H100s respectively.

Figure 1: Standard speculative decoding versus our judge decoding strategy for `Llama-3.1-8B` as draft and `Llama-3.1-405B` as target. Accepted (rejected) tokens are highlighted in green (red).

[*]Work done during an internship at Meta GenAI. [†]Meta GenAI. [‡]ETH Zürich.
Correspondence to: gregorb@ethz.ch

LoRA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

Edward Hu[*] Yelong Shen[*] Phillip Wallis Zeyuan Allen-Zhu
Yuanzhi Li Shean Wang Lu Wang Weizhu Chen
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

ABSTRACT

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Low-Rank Adaptation**, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at `https://github.com/microsoft/LoRA`.

1 INTRODUCTION

Many applications in natural language processing rely on adapting *one* large-scale, pre-trained language model to *multiple* downstream applications. Such adaptation is usually done via *fine-tuning*, which updates all the parameters of the pre-trained model. The major downside of fine-tuning is that the new model contains as many parameters as in the original model. As larger models are trained every few months, this changes from a mere "inconvenience" for GPT-2 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a critical deployment challenge for GPT-3 (Brown et al., 2020) with 175 billion trainable parameters.[1]

Many sought to mitigate this by adapting only some parameters or learning external modules for new tasks. This way, we only need to store and load a small number of task-specific parameters in addition to the pre-trained model for each task, greatly boosting the operational efficiency when deployed. However, existing techniques

Figure 1: Our reparametrization. We only train A and B.

[*]Equal contribution.
[3]Compared to V1, this draft includes better baselines, experiments on GLUE, and more on adapter latency.
[1]While GPT-3 175B achieves non-trivial performance with few-shot learning, fine-tuning boosts its performance significantly as shown in Appendix A.

Speculative decoding and LLM-as-a-judge

Today

After: Fourth batch of four **paper presentations**.

Adding Conditional Control to Text-to-Image Diffusion Models

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala
Stanford University
{lvmin, anyirao, maneesh}@cs.stanford.edu

Input Canny edge | Default | "masterpiece of fairy tale, giant deer, golden antlers" | "..., quaint city Galic"

Input human pose | Default | "chef in kitchen" | "Lincoln statue"

Figure 1: Controlling Stable Diffusion with learned conditions. ControlNet allows users to add conditions like Canny edges (top), human pose (bottom), *etc.*, to control the image generation of large pretrained diffusion models. The default results use the prompt "a high-quality, detailed, and professional image". Users can optionally give prompts like the "chef in kitchen".

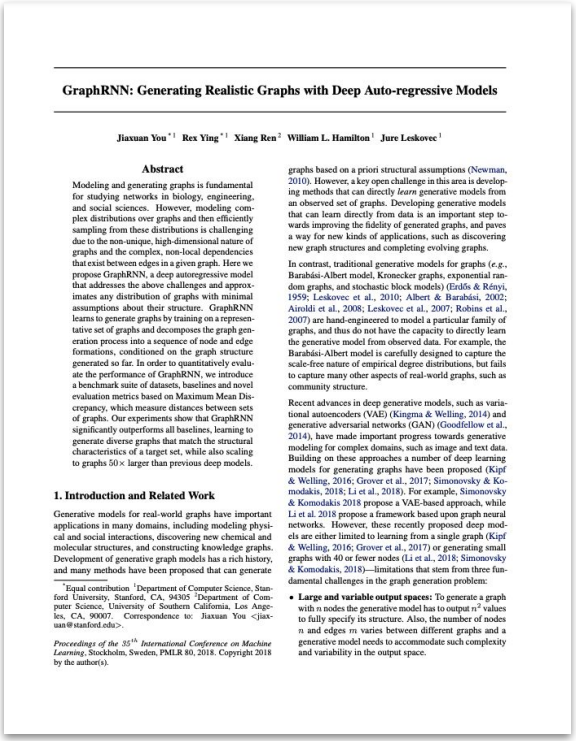
Abstract

We present ControlNet, a neural network architecture to add spatial conditioning controls to large, pretrained text-to-image diffusion models. ControlNet locks the production-ready large diffusion models, and reuses their deep and robust encoding layers pretrained with billions of images as a strong backbone to learn a diverse set of conditional controls. The neural architecture is connected with "zero convolutions" (zero-initialized convolution layers) that progressively grow the parameters from zero and ensure that no harmful noise could affect the finetuning. We test various conditioning controls, e.g., edges, depth, segmentation, human pose, etc., with Stable Diffusion, using single or multiple conditions, with or without prompts. We show that the training of ControlNets is robust with small (<50k) and large (>1m) datasets. Extensive results show that ControlNet may facilitate wider applications to control image diffusion models.

1. Introduction

Many of us have experienced flashes of visual inspiration that we wish to capture in a unique image. With the advent of text-to-image diffusion models [54, 62, 72], we can now create visually stunning images by typing in a text prompt. Yet, text-to-image models are limited in the control they provide over the spatial composition of the image: precisely expressing complex layouts, poses, shapes and forms can be difficult via text prompts alone. Generating an image that accurately matches our mental imagery often requires numerous trial-and-error cycles of editing a prompt, inspecting the resulting images and then re-editing the prompt.

Can we enable finer grained spatial control by letting users provide additional images that directly specify their desired image composition? In computer vision and machine learning, these additional images (*e.g.*, edge maps, human pose skeletons, segmentation maps, depth, normals, *etc.*) are often treated as conditioning on the image generation process. Image-to-image translation models [34, 98] learn



GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models

Jiaxuan You [*1] Rex Ying [*1] Xiang Ren [2] William L. Hamilton [1] Jure Leskovec [1]

Abstract

Modeling and generating graphs is fundamental for studying networks in biology, engineering, and social sciences. However, modeling complex distributions over graphs and then efficiently sampling from these distributions is challenging due to the non-unique, high-dimensional nature of graphs and the complex, non-local dependencies that exist between edges in a given graph. Here we propose GraphRNN, a deep autoregressive model that addresses the above challenges and approximates any distribution of graphs with minimal assumptions about their structure. GraphRNN learns to generate graphs by training on a representative set of graphs and decomposes the graph generation process into a sequence of node and edge formations, conditioned on the graph structure generated so far. In order to quantitatively evaluate the performance of GraphRNN, we introduce a benchmark suite of datasets, baselines and novel evaluation metrics based on Maximum Mean Discrepancy, which measure distances between sets of graphs. Our experiments show that GraphRNN significantly outperforms all baselines, learning to generate diverse graphs that match the structural characteristics of a target set, while also scaling to graphs $50\times$ larger than previous deep models.

1. Introduction and Related Work

Generative models for real-world graphs have important applications in many domains, including modeling physical and social interactions, discovering new chemical and molecular structures, and constructing knowledge graphs. Development of generative graph models has a rich history, and many methods have been proposed that can generate graphs based on a priori structural assumptions (Newman, 2010). However, a key open challenge in this area is developing methods that can directly *learn* generative models from an observed set of graphs. Developing generative models that can learn directly from data is an important step towards improving the fidelity of generated graphs, and paves a way for new kinds of applications, such as discovering new graph structures and completing evolving graphs.

In contrast, traditional generative models for graphs (*e.g.*, Barabási-Albert model, Kronecker graphs, exponential random graphs, and stochastic block models) (Erdős & Rényi, 1959; Leskovec et al., 2010; Albert & Barabási, 2002; Airoldi et al., 2008; Leskovec et al., 2007; Robins et al., 2007) are hand-engineered to model a particular family of graphs, and thus do not have the capacity to directly learn the generative model from observed data. For example, the Barabási-Albert model is carefully designed to capture the scale-free nature of empirical degree distributions, but fails to capture many other aspects of real-world graphs, such as community structure.

Recent advances in deep generative models, such as variational autoencoders (VAE) (Kingma & Welling, 2014) and generative adversarial networks (GAN) (Goodfellow et al., 2014), have made important progress towards generative modeling for complex domains, such as image and text data. Building on these approaches a number of deep learning models for generating graphs have been proposed (Kipf & Welling, 2016; Grover et al., 2017; Simonovsky & Komodakis, 2018; Li et al., 2018). For example, Simonovsky & Komodakis 2018 propose a VAE-based approach, while Li et al. 2018 propose a framework based upon graph neural networks. However, these recently proposed deep models are either limited to learning from a single graph (Kipf & Welling, 2016; Grover et al., 2017) or generating small graphs with 40 or fewer nodes (Li et al., 2018; Simonovsky & Komodakis, 2018)—limitations that stem from three fundamental challenges in the graph generation problem:

- **Large and variable output spaces:** To generate a graph with n nodes the generative model has to output n^2 values to fully specify its structure. Also, the number of nodes n and edges m varies between different graphs and a generative model needs to accommodate such complexity and variability in the output space.

[*]Equal contribution [1]Department of Computer Science, Stanford University, Stanford, CA, 94305 [2]Department of Computer Science, University of Southern California, Los Angeles, CA, 90007. Correspondence to: Jiaxuan You <jiaxuan@stanford.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

Published as a conference paper at ICLR 2025

JUDGE DECODING: FASTER SPECULATIVE SAMPLING REQUIRES GOING BEYOND MODEL ALIGNMENT

Gregor Bachmann[†,♯,*] Sotiris Anagnostidis[†,♯] Albert Pumarola[†]
Markos Georgopoulos[†] Artsiom Sanakoyeu[†] Yuming Du[†]
Edgar Schönfeld[†] Ali Thabet[†] Jonas Kohler[†]

ABSTRACT

The performance of large language models (LLMs) is closely linked to their underlying size, leading to ever-growing networks and hence slower inference. Speculative decoding has been proposed as a technique to accelerate autoregressive generation, leveraging a fast draft model to propose candidate tokens, which are then verified in parallel based on their likelihood under the target model. While this approach guarantees to reproduce the target output, it incurs a substantial penalty: many high-quality draft tokens are rejected, even when they represent objectively valid continuations. Indeed, we show that even powerful draft models such as `GPT-4o`, as well as human text cannot achieve high acceptance rates under the standard verification scheme. This severely limits the speedup potential of current speculative decoding methods, as an early rejection becomes overwhelmingly likely when solely relying on alignment of draft and target.
We thus ask the following question: *Can we adapt verification to recognize correct, but non-aligned replies?* To this end, we draw inspiration from the *LLM-as-a-judge* framework, which demonstrated that LLMs are able to rate answers in a versatile way. We carefully design a dataset to elicit the same capability in the target model by training a compact module on top of the embeddings to produce "judgements" of the current continuation. We showcase our strategy on the `Llama-3.1` family, where our `8b/405B-Judge` achieves a speedup of $9\times$ over `Llama-405B`, while maintaining its quality on a large range of benchmarks. These benefits remain present even in optimized inference frameworks, where our method reaches up to 141 tokens/s for `8B/70B-Judge` and 129 tokens/s for `8B/405B` on 2 and 8 H100s respectively.

Figure 1: Standard speculative decoding versus our judge decoding strategy for `Llama-3.1-8B` as draft and `Llama-3.1-405B` as target. Accepted (rejected) tokens are highlighted in green (red).

[*]Work done during an internship at Meta GenAI. [†]Meta GenAI. [♯]ETH Zürich.
Correspondence to: gregorb@ethz.ch

LoRA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

Edward Hu* Yelong Shen* Phillip Wallis Zeyuan Allen-Zhu
Yuanzhi Li Shean Wang Lu Wang Weizhu Chen
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

ABSTRACT

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Low-Rank Adaptation**, or **LoRA**, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at `https://github.com/microsoft/LoRA`.

1 INTRODUCTION

Many applications in natural language processing rely on adapting *one* large-scale, pre-trained language model to *multiple* downstream applications. Such adaptation is usually done via *fine-tuning*, which updates all the parameters of the pre-trained model. The major downside of fine-tuning is that the new model contains as many parameters as in the original model. As larger models are trained every few months, this changes from a mere "inconvenience" for GPT-2 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a critical deployment challenge for GPT-3 (Brown et al., 2020) with 175 billion trainable parameters.[1]

Many sought to mitigate this by adapting only some parameters or learning external modules for new tasks. This way, we only need to store and load a small number of task-specific parameters in addition to the pre-trained model for each task, greatly boosting the operational efficiency when deployed. However, existing techniques

Figure 1: Our reparametrization. We only train A and B.

[*]Equal contribution.
[3]Compared to V1, this draft includes better baselines, experiments on GLUE, and more on adapter latency.
[1]While GPT-3 175B achieves non-trivial performance with few-shot learning, fine-tuning boosts its performance significantly as shown in Appendix A.

LoRA — very popular efficient fine-tuning method

Quick Aside

Midway Reports and Grading:

- Thanks everyone for submitting your midway reports!
- We are going through these, and aim to give feedback / grades next week.

Last Class: Final DGM — connections between: Score-based generative models, diffusion models, and VAEs

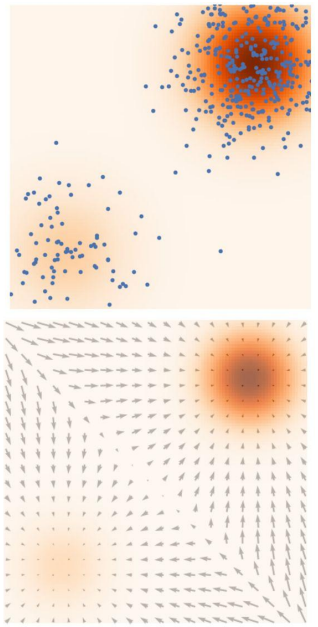
Review — Finished Score Matching

We learned about:

Review — Finished Score Matching

We learned about:

- Score-based generative models via *score matching* and *Langevin Monte Carlo*.
  - Learned by optimizing the Fisher divergence objective.
- Difficulties with sample coverages and score estimation.
- Extensions to noise-conditional score matching (NCSM).
  - Objective: weighted sum of Fisher divergences.
- Sampling via annealed LMC over multiple noise levels.

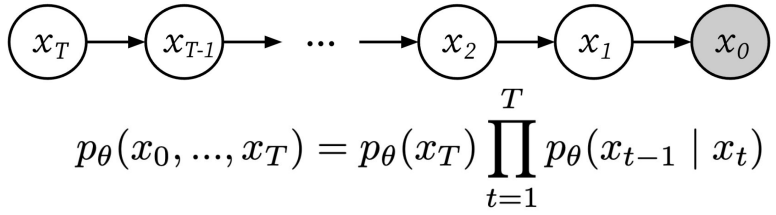


Review — Connections to DDPMs & VAEs

Review — Connections to DDPMs & VAEs

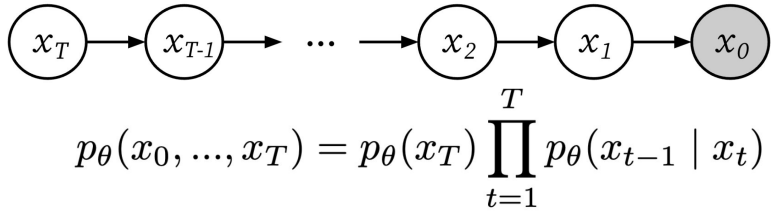
(1) DDPM shows that under a certain parameterization of the model, their algorithm is *equivalent to* denosing score matching.

Review — Connections to DDPMs & VAEs



(1) DDPM shows that under a certain parameterization of the model, their algorithm is *equivalent to* denosing score matching.

Review — Connections to DDPMs & VAEs



(1) DDPM shows that under a certain parameterization of the model, their algorithm is *equivalent to* denosing score matching.

- Shows optimizing the ELBO ⇒ equivalent to the "*sum of Fisher divergences*" loss from before!
- And that sampling from the forward model ⇒ equivalent to Langevin dynamics on a learned score function!

Review — Connections to DDPMs & VAEs

DDPM Samples

(1) DDPM shows that under a certain parameterization of the model, their algorithm is *equivalent to* denosing score matching.

- Shows optimizing the ELBO ⇒ equivalent to the "*sum of Fisher divergences*" loss from before!
- And that sampling from the forward model ⇒ equivalent to Langevin dynamics on a learned score function!

(2) Shows that under this parameterization (along with a few additional tricks to better handle image data), sample quality is can be very good.

(Possibly better than most/all previous generative models up until this point).

Review — Explicit vs Implicit Probabilistic Generative Models

Review — Explicit vs Implicit Probabilistic Generative Models

Explicit

Defines a probability model

Then learns parameter of model.

Review — Explicit vs Implicit Probabilistic Generative Models

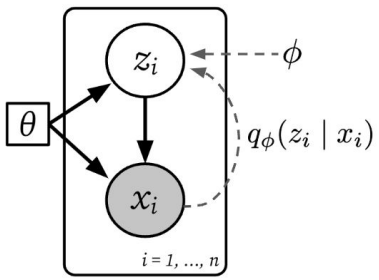
<u>Explicit</u>

Defines a probability model

Then learns parameter of model.

E.g.,

VAEs



Review — Explicit vs Implicit Probabilistic Generative Models

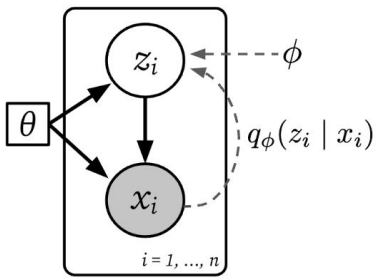
<u>Explicit</u>

Defines a probability model

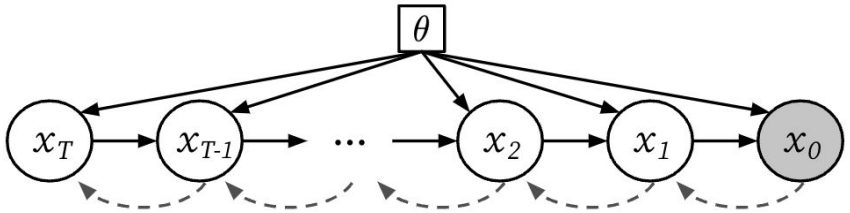
Then learns parameter of model.

E.g.,

VAEs



Diffusion Models



Review — Explicit vs Implicit Probabilistic Generative Models

Explicit

Defines a probability model

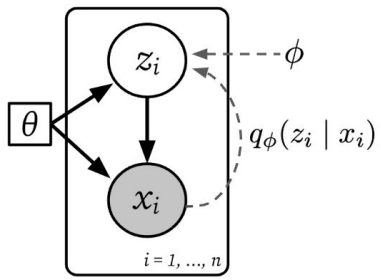
Then learns parameter of model.

Implicit

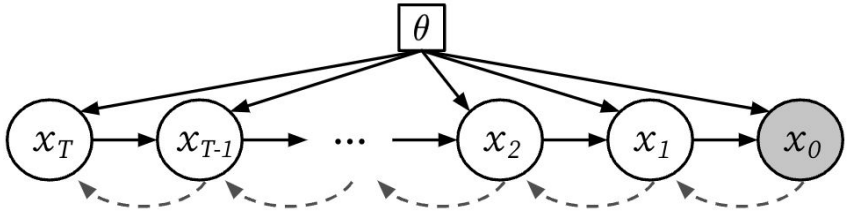
Learns some quantity that implies a probabilistic model and allows for sampling…

E.g.,

VAEs



Diffusion Models



Review — Explicit vs Implicit Probabilistic Generative Models

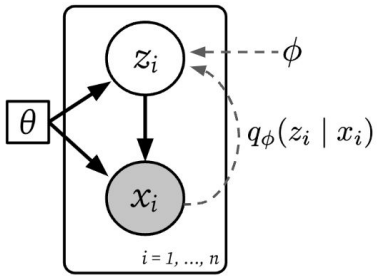
Explicit

Defines a probability model

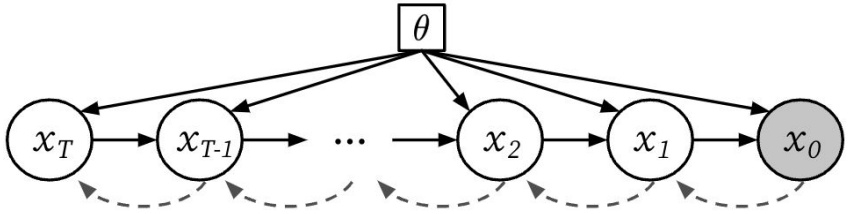
Then learns parameter of model.

E.g.,

VAEs



Diffusion Models



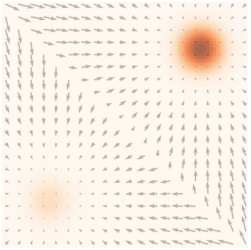
Implicit

Learns some quantity that implies a probabilistic model and allows for sampling…

E.g.,

Score-based Models

$$s(x) = \nabla_x \log p(x)$$



Source: Yang Song

Review — Explicit vs Implicit Probabilistic Generative Models

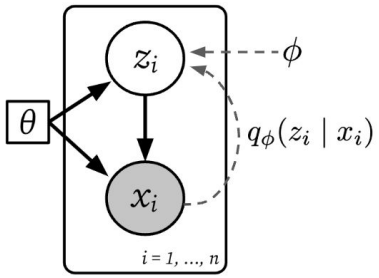
Explicit

Defines a probability model

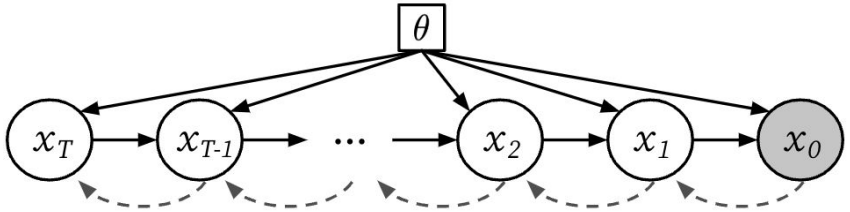
Then learns parameter of model.

E.g.,

VAEs



Diffusion Models



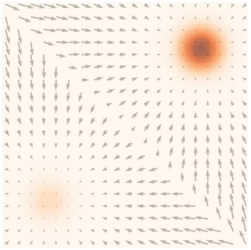
Implicit

Learns some quantity that implies a probabilistic model and allows for sampling…

E.g.,

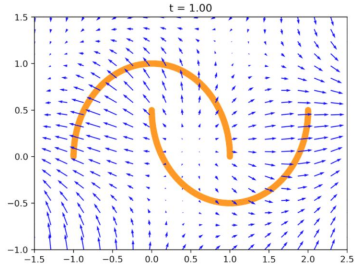
Score-based Models

$$s(x) = \nabla_x \log p(x)$$



Source: Yang Song

Flow-matching



Velocity field

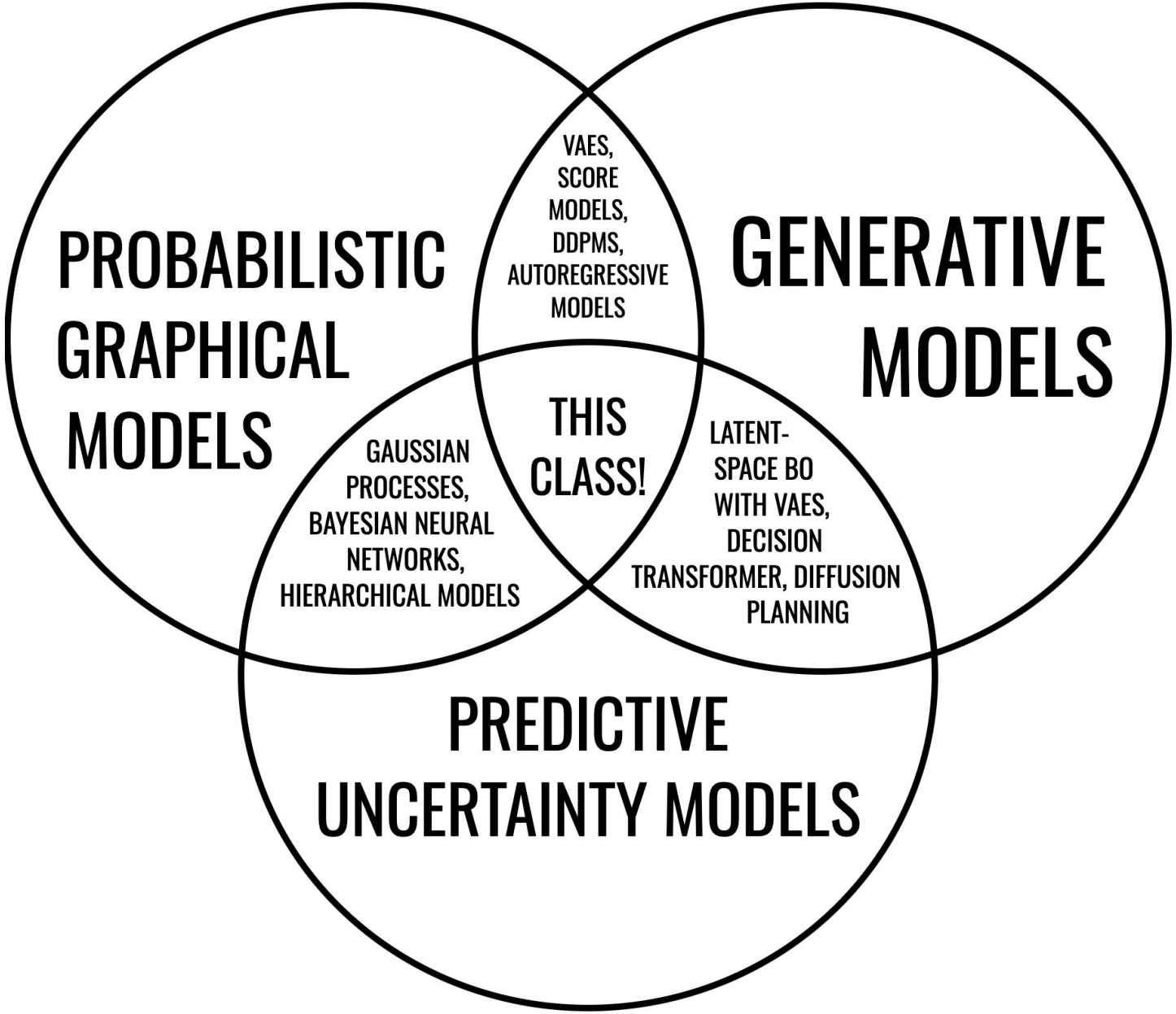
Source: Jakub Tomczak

Predictive Uncertainty Quantification

Class Outline

This course focuses on probabilistic models and their central role within modern machine learning and generative modeling.

This course is inspired by some previous classes…



Deep Generative Models vs Predictive UQ Models

At a high level…

Deep Generative Models vs Predictive UQ Models

At a high level…

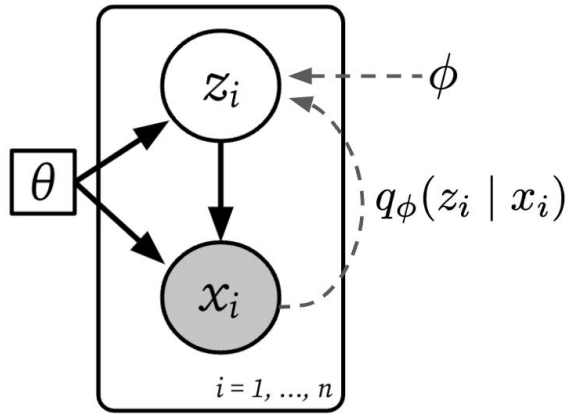
We have been focusing on deep generative (probabilistic) models.

Deep Generative Models vs Predictive UQ Models

At a high level…

We have been focusing on deep generative (probabilistic) models.

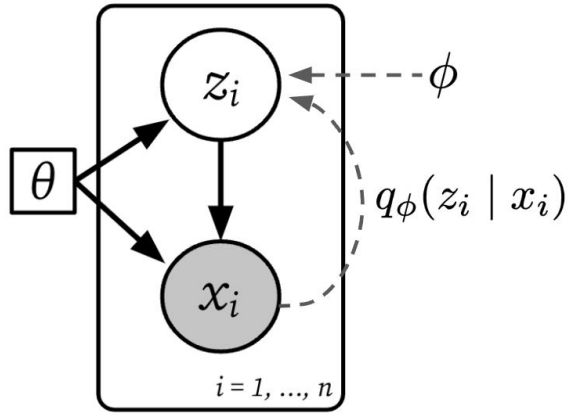
Typically: defined as a joint PDF over observations and often latent variables — which we can describe via PGMs.



Deep Generative Models vs Predictive UQ Models

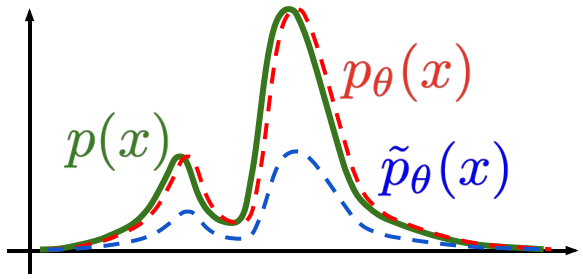
At a high level…

We have been focusing on deep generative (probabilistic) models.



Typically: defined as a joint PDF over observations and often latent variables — which we can describe via PGMs.

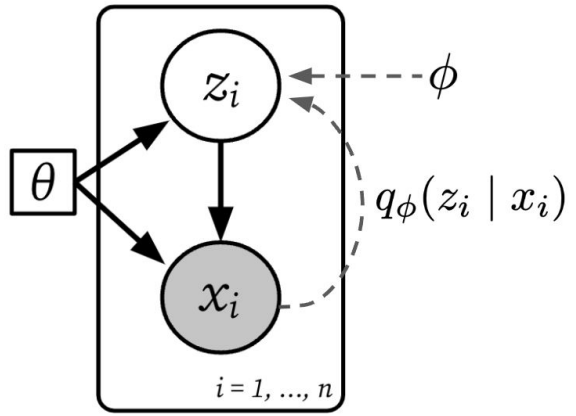
And we discussed (approximate) inference algorithms and/or learning (parameter estimation) algorithms in these models.



Deep Generative Models vs Predictive UQ Models

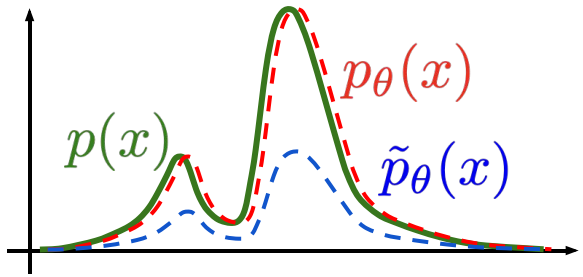
At a high level…

We have been focusing on deep generative (probabilistic) models.



Typically: defined as a joint PDF over observations and often latent variables — which we can describe via PGMs.

And we discussed (approximate) inference algorithms and/or learning (parameter estimation) algorithms in these models.



And this was used in many tasks, *e.g.*, inference queries and generative modeling.

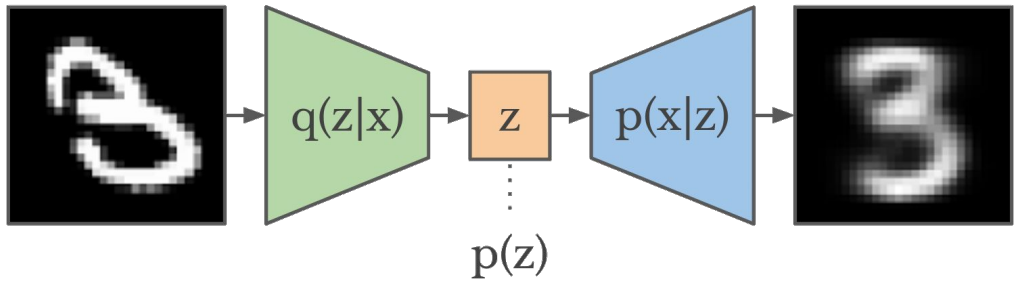
Deep Generative Models vs Predictive UQ Models

Technically: these tasks can often be viewed as **unsupervised learning**.

Deep Generative Models vs Predictive UQ Models

Technically: these tasks can often be viewed as **unsupervised learning**.

(Though often supervised learning techniques are used — overall might be viewed as *self-supervised learning*)

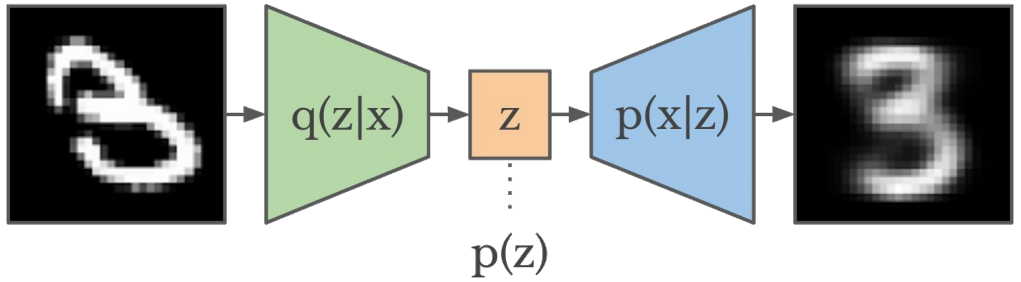


"Building Variational Auto-Encoders in TensorFlow", Danijar Hafner

Deep Generative Models vs Predictive UQ Models

Technically: these tasks can often be viewed as **unsupervised learning**.

(Though often supervised learning techniques are used — overall might be viewed as *self-supervised learning*)



"Building Variational Auto-Encoders in TensorFlow", Danijar Hafner

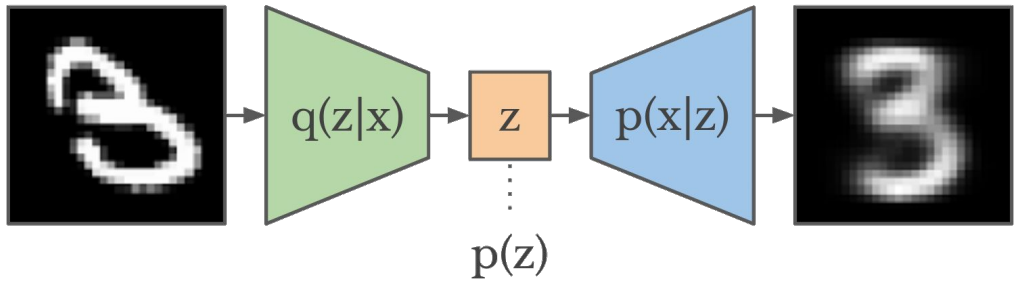
Where, to be clear, supervised learning is when:

- We have a labeled dataset of (x, y) pairs.
- Try to learn a function mapping x to y.

Deep Generative Models vs Predictive UQ Models

Technically: these tasks can often be viewed as **unsupervised learning**.

(Though often supervised learning techniques are used — overall might be viewed as *self-supervised learning*)



"Building Variational Auto-Encoders in TensorFlow", Danijar Hafner

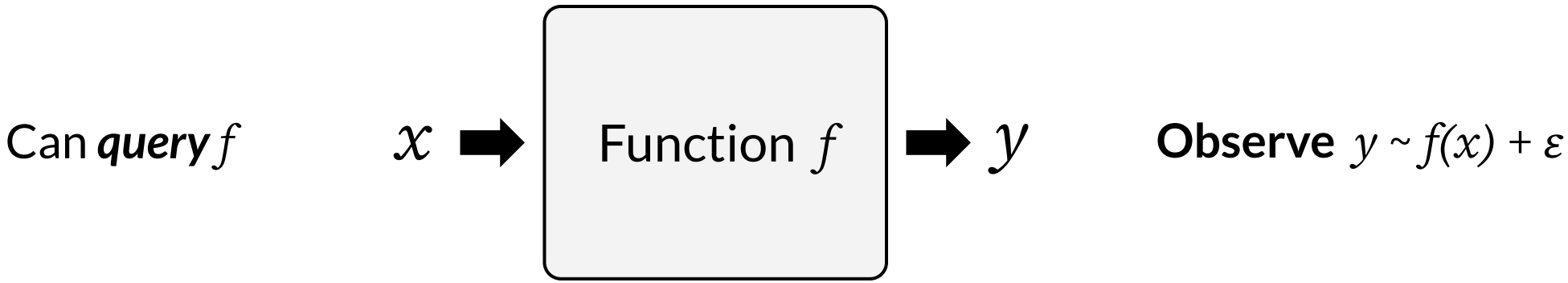
Where, to be clear, supervised learning is when:

- We have a labeled dataset of (x, y) pairs.
- Try to learn a function mapping x to y.

In this lecture we will be focusing on probabilistic models for supervised learning.

Deep Generative Models vs Predictive UQ Models

More formally, suppose that we have a (possibly noisy) function f, from which we observe many input/output (x, y) pairs.



Deep Generative Models vs Predictive UQ Models

More formally, suppose that we have a (possibly noisy) function f, from which we observe many input/output (x, y) pairs.

Can ***query*** f x ➡ Function f ➡ y **Observe** $y \sim f(x) + \varepsilon$

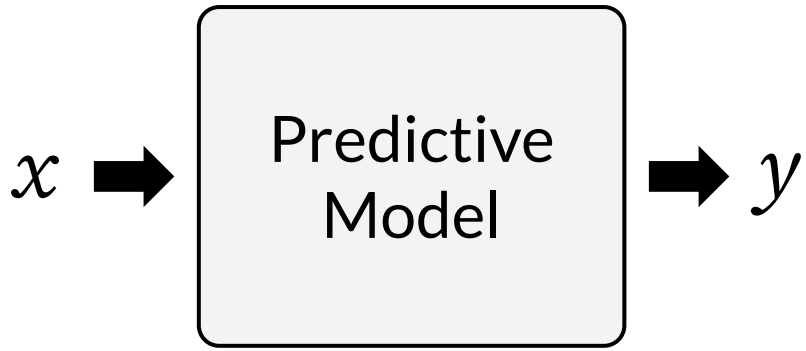
And let's suppose we have a dataset of n (x, y) pairs: $D_n = \{ [x_1, y_1], [x_2, y_2], \vdots, [x_n, y_n] \}$

Deep Generative Models vs Predictive UQ Models

In predictive models, given an input x, we aim to make a point-prediction of y.
(*E.g.*, could be either regression or classification, depending on y).

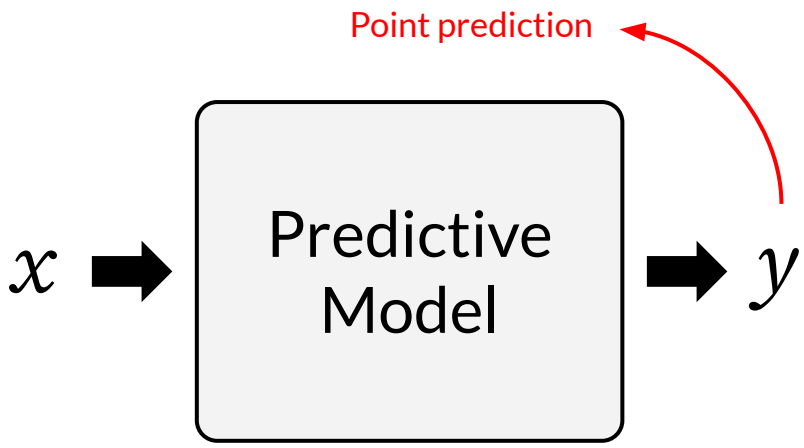
Deep Generative Models vs Predictive UQ Models

In predictive models, given an input x, we aim to make a point-prediction of y.
(*E.g.*, could be either regression or classification, depending on y).



Deep Generative Models vs Predictive UQ Models

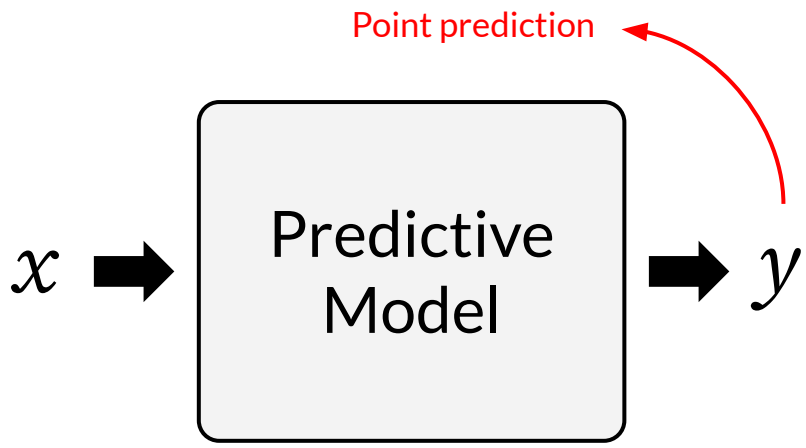
In predictive models, given an input x, we aim to make a point-prediction of y.
(*E.g.*, could be either regression or classification, depending on y).



Deep Generative Models vs Predictive UQ Models

In predictive models, given an input x, we aim to make a point-prediction of y.
(*E.g.*, could be either regression or classification, depending on y).

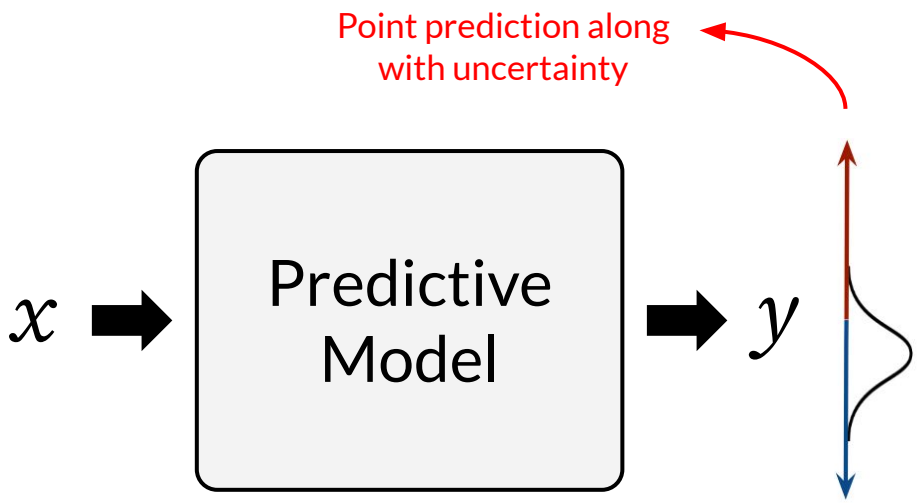
In predictive UQ, we instead aim to make a prediction along with some measure of uncertainty about that prediction.



Deep Generative Models vs Predictive UQ Models

In predictive models, given an input x, we aim to make a point-prediction of y.
(*E.g.*, could be either regression or classification, depending on y).

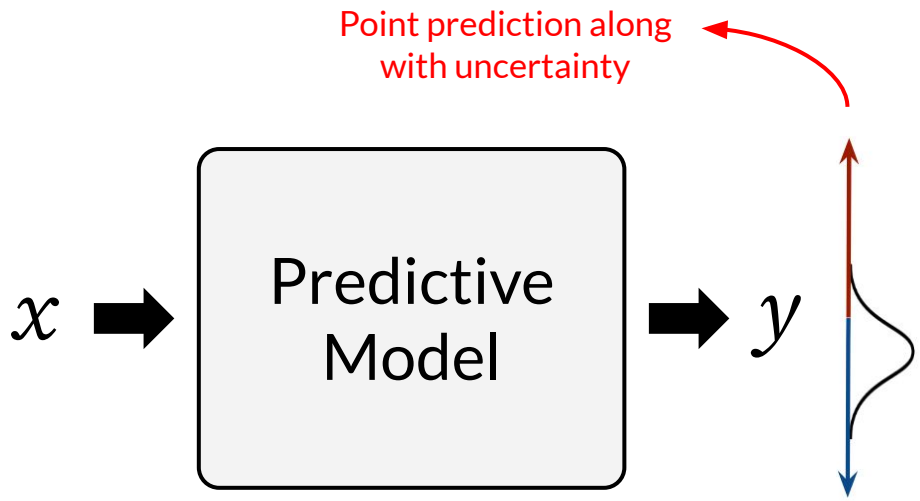
In predictive UQ, we instead aim to make a prediction along with some measure of uncertainty about that prediction.

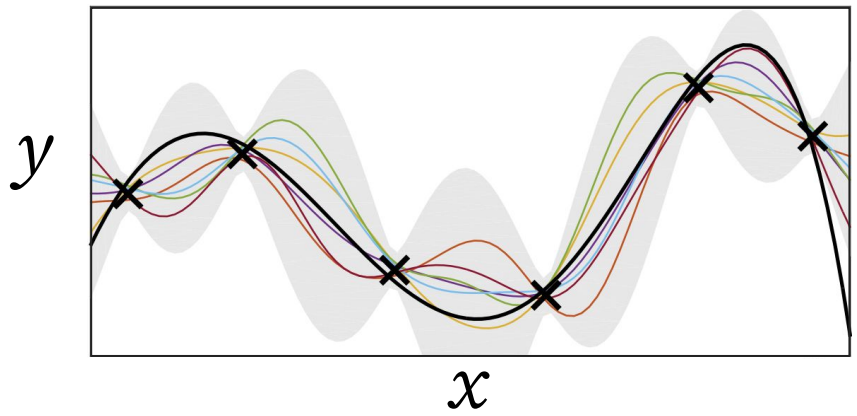


Deep Generative Models vs Predictive UQ Models

In predictive models, given an input x, we aim to make a point-prediction of y.
(*E.g.*, could be either regression or classification, depending on y).

In predictive UQ, we instead aim to make a prediction along with some measure of uncertainty about that prediction.

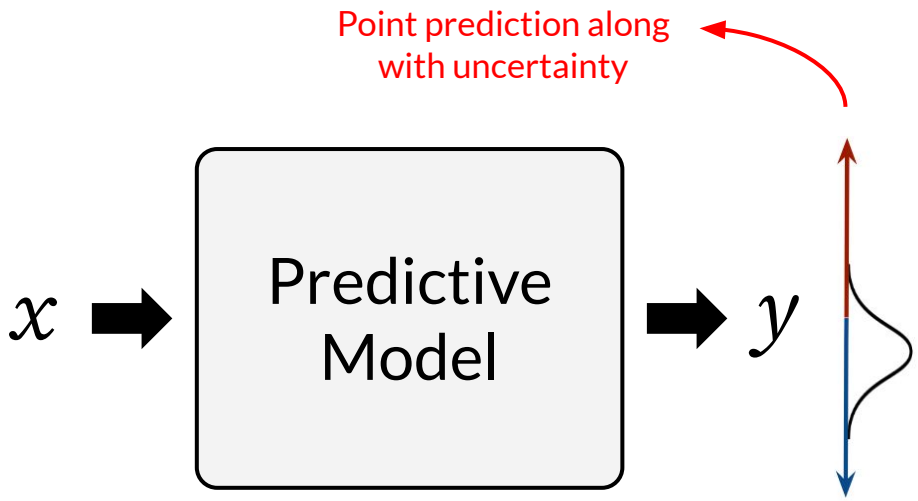


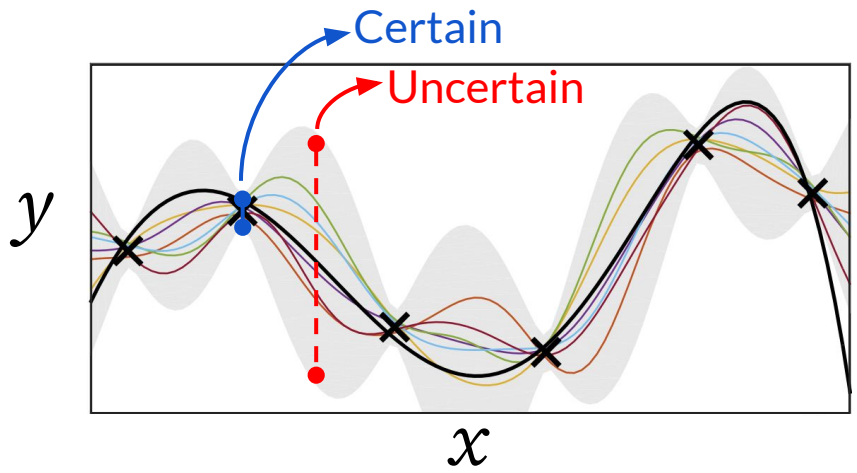


Deep Generative Models vs Predictive UQ Models

In predictive models, given an input x, we aim to make a point-prediction of y.
(*E.g.*, could be either regression or classification, depending on y).

In predictive UQ, we instead aim to make a prediction along with some measure of uncertainty about that prediction.





Predictive UQ Models — Motivation

Predictions **with uncertainty** are useful! For a few reasons:

Predictive UQ Models — Motivation

Predictions **with uncertainty** are useful! For a few reasons:

(1) Allows one to see *multiple plausible predictions* (rather than just one) — sometimes as an interval/region, a set, a probability distribution, samples, etc.

Predictive UQ Models — Motivation

Predictions **with uncertainty** are useful! For a few reasons:

(1) Allows one to see *multiple plausible predictions* (rather than just one) — sometimes as an interval/region, a set, a probability distribution, samples, etc.

(2) Can yield ML methods that *abstain* when they are not confident enough.

Predictive UQ Models — Motivation

Predictions **with uncertainty** are useful! For a few reasons:

(1) Allows one to see *multiple plausible predictions* (rather than just one) — sometimes as an interval/region, a set, a probability distribution, samples, etc.

(2) Can yield ML methods that *abstain* when they are not confident enough.

(3) Relatedly: this leads to *trustworthy AI* methods, since you know when to use and/or ignore predictions (*e.g.*, in high-risk areas, like medicine, law & justice, finance, etc).

Predictive UQ Models — Motivation

Predictions **with uncertainty** are useful! For a few reasons:

(1) Allows one to see *multiple plausible predictions* (rather than just one) — sometimes as an interval/region, a set, a probability distribution, samples, etc.

(2) Can yield ML methods that *abstain* when they are not confident enough.

(3) Relatedly: this leads to *trustworthy AI* methods, since you know when to use and/or ignore predictions (*e.g.*, in high-risk areas, like medicine, law & justice, finance, etc).

(4) Allows for better *ML + human interaction* and *automation* — we can estimate when AI should do a task and when a human should take over.

Predictive UQ Models — Motivation

Predictions **with uncertainty** are useful! For a few reasons:

(5) Sequential decision making under uncertainty → *an important one (in my opinion).*

Predictive uncertainty models allow for optimal decision making under uncertainty.

Predictive UQ Models — Motivation

Predictions **with uncertainty** are useful! For a few reasons:

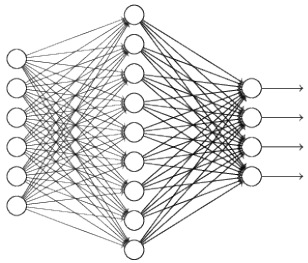
(5) Sequential decision making under uncertainty → *an important one (in my opinion).*

Predictive uncertainty models allow for optimal decision making under uncertainty.

Let me give a high-level intuition for how predictive UQ is used in decision making:

Predictive UQ Models — Motivation

Machine learning is based on predictions — typically point predictions.

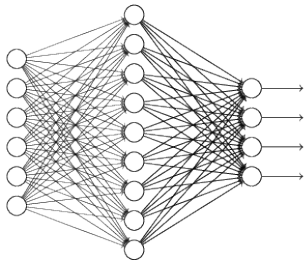


Monkey

Predictive UQ Models — Motivation

Machine learning is based on predictions — typically point predictions.

Suppose we augment these predictions **with reliable uncertainties** (*e.g.*, with a measure of confidence) …



Monkey?

Cat?

Rabid Cat?

Cotton Ball?

Porcupine?

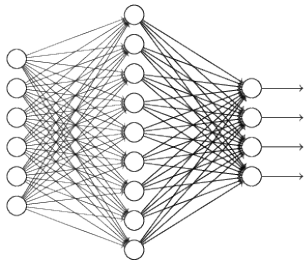
Predictive UQ Models — Motivation

Machine learning is based on predictions — typically point predictions.

Suppose we augment these predictions **with reliable uncertainties** (*e.g.*, with a measure of confidence) …

We can then balance our uncertainty about possible outcomes to make optimal choices (aka ⇒ *decision-making under uncertainty*).

This applies to both classical probabilistic models and probabilistic deep learning models.



Monkey?

Cat?

Rabid Cat?

Cotton Ball?

Porcupine?

"Should I pet it?"

Predictive UQ Models —Decision Making Under Uncertainty

Many applications of these models in decision making under uncertainty.

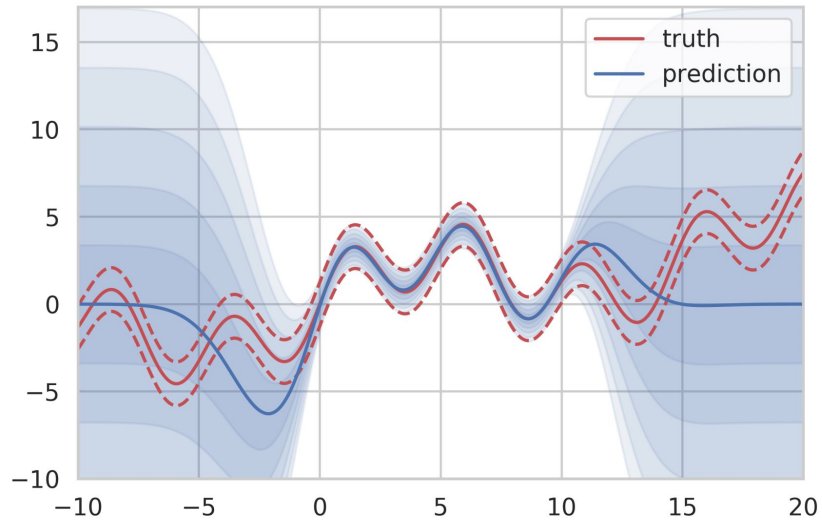
(Especially in the sciences and engineering).

Predictive UQ Models —Decision Making Under Uncertainty

Many applications of these models in decision making under uncertainty.

(Especially in the sciences and engineering).

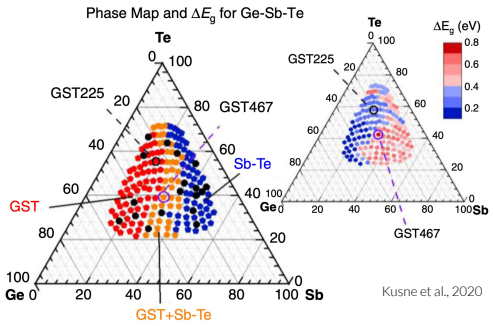
E.g., using probabilistic models for optimization, experimental design, and active learning.



Predictive UQ Models —Decision Making Under Uncertainty

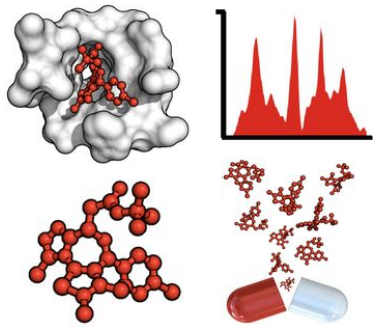
SCIENCE

Materials design



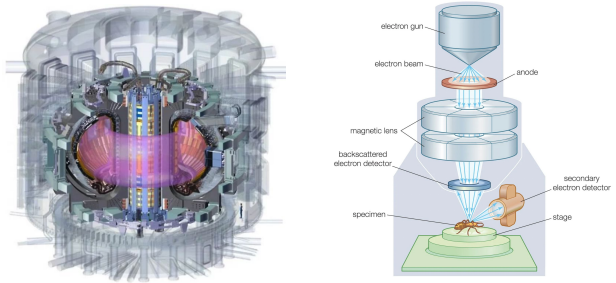
Materials design over large spaces of compounds.

Drug discovery



Target identification, lead optimization.

Large Science Machines



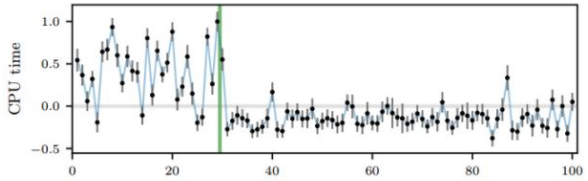
Optimize settings & controls for science use-cases.

[Kusne et al., 2020], [Eisenstein, 2020], [Fenstermacher et al., 2022]

Predictive UQ Models —Decision Making Under Uncertainty

ENGINEERING

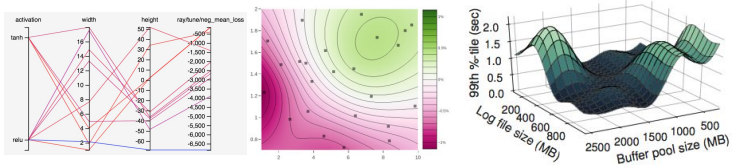
Computer Systems



OTTER TUNE

Monitor and tune configurations for performance (latency, throughput).

Machine Learning



Find and assess new methods, models, architectures, hyperparameters.

[Van Aken et al, 2017], [Balandat et al., 2020]

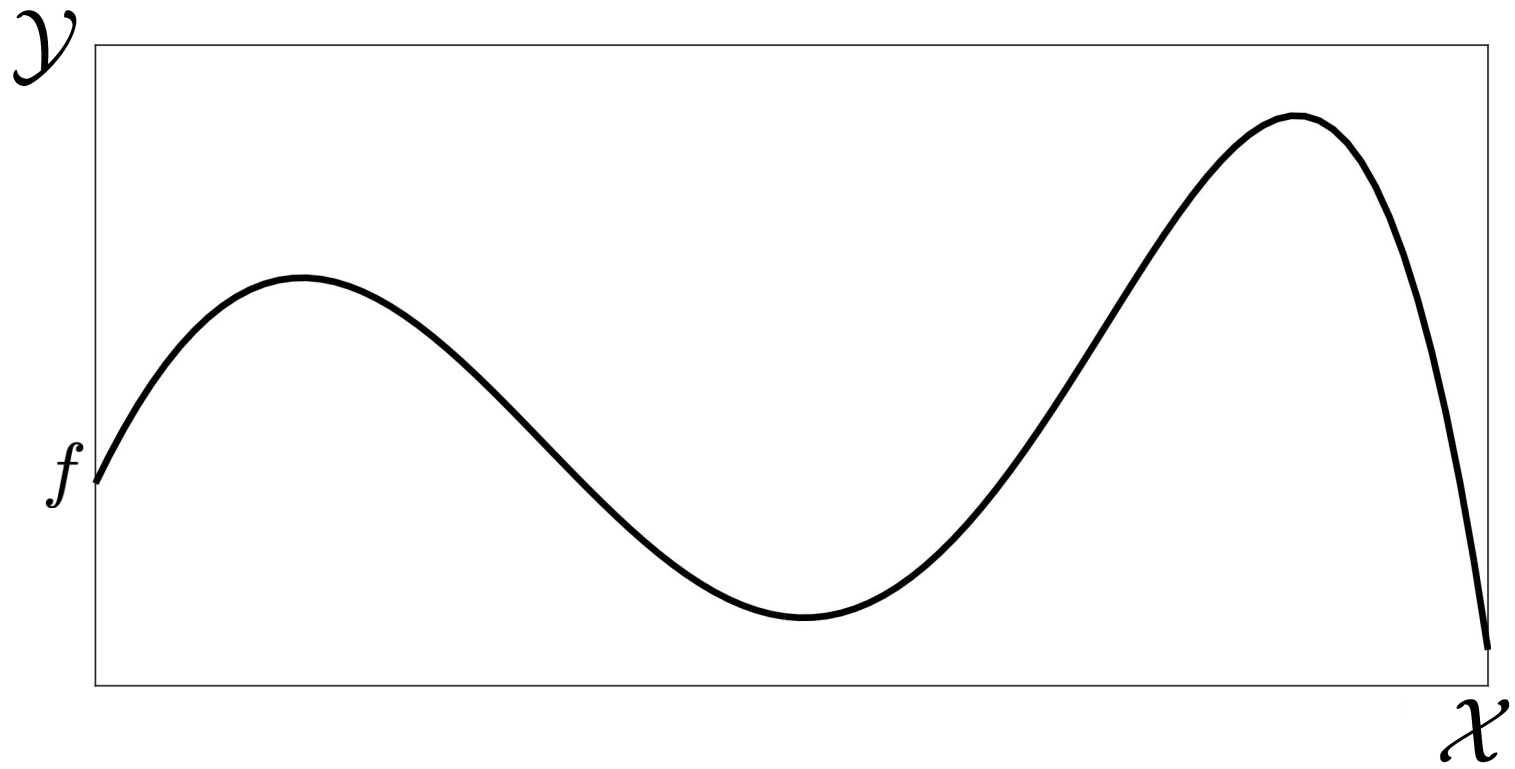
Predictive UQ Models — Background and Typical Setup

Predictive UQ Models — Background and Typical Setup

Suppose we have an input space (*domain*) $\mathcal{X}$, output space $\mathcal{Y}$, and a function f.

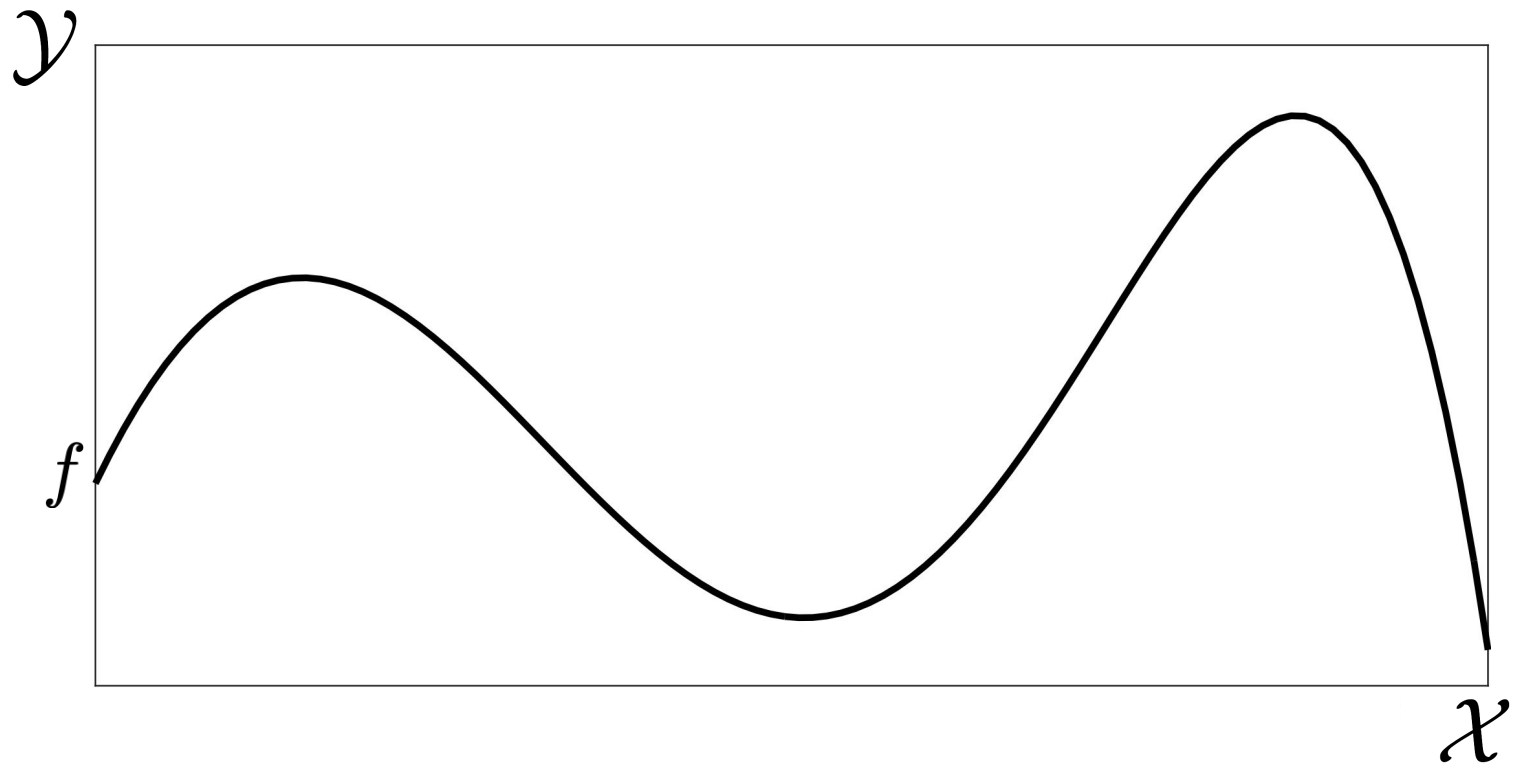
Predictive UQ Models — Background and Typical Setup

Suppose we have an input space (*domain*) $\mathcal{X}$, output space $\mathcal{Y}$, and a function f.



Predictive UQ Models — Background and Typical Setup

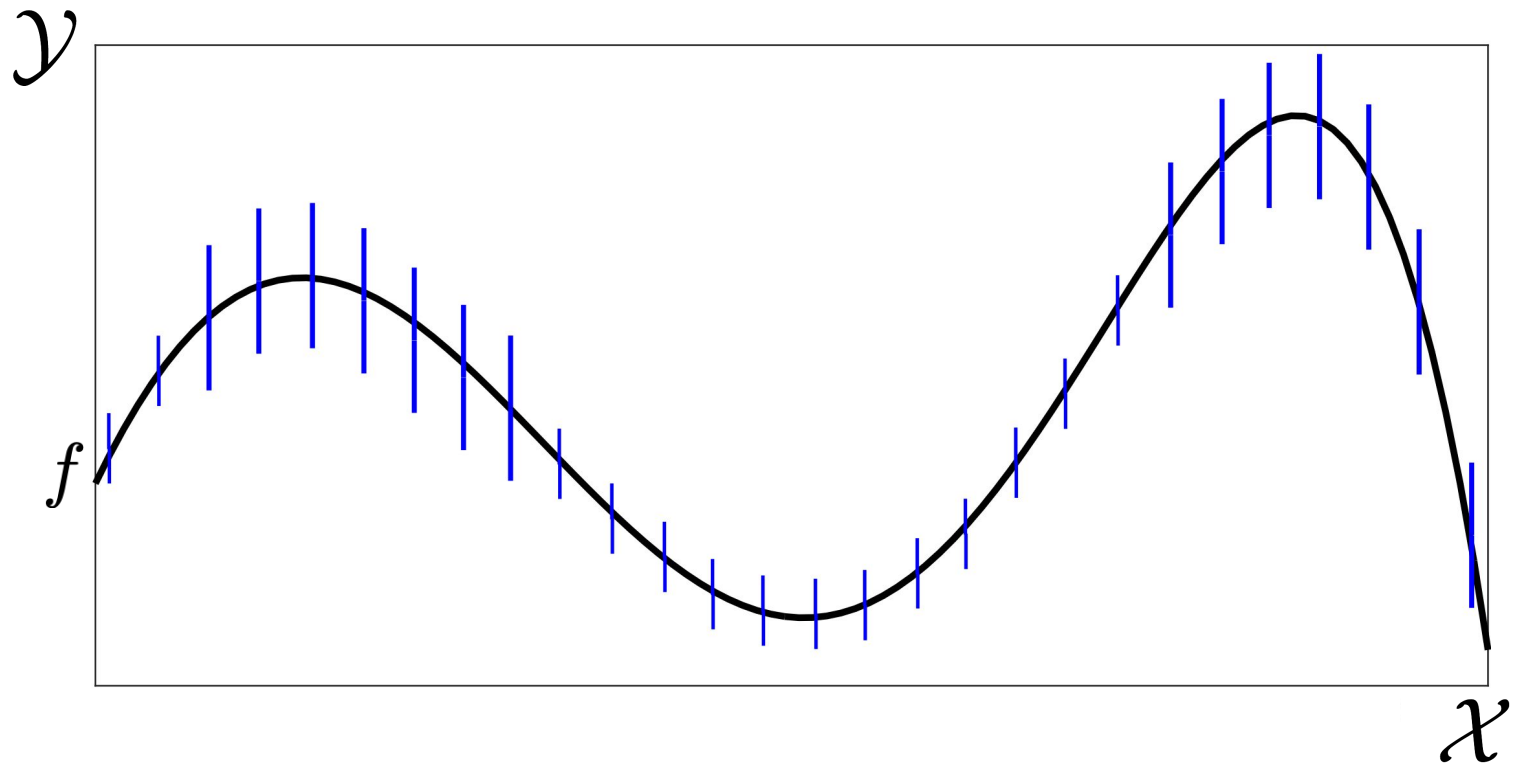
It is a **noisy function** ⇒ for each $x \in \mathcal{X}$, there is a distribution over $y \in \mathcal{Y}$.



Predictive UQ Models — Background and Typical Setup

It is a **noisy function** ⇒ for each $x \in \mathcal{X}$, there is a distribution over $y \in \mathcal{Y}$.

Shown **in blue** below.



Predictive UQ Models — Background and Typical Setup

It is a **noisy function** ⇒ for each $x \in \mathcal{X}$, there is a distribution over $y \in \mathcal{Y}$.

Shown **in blue** below.

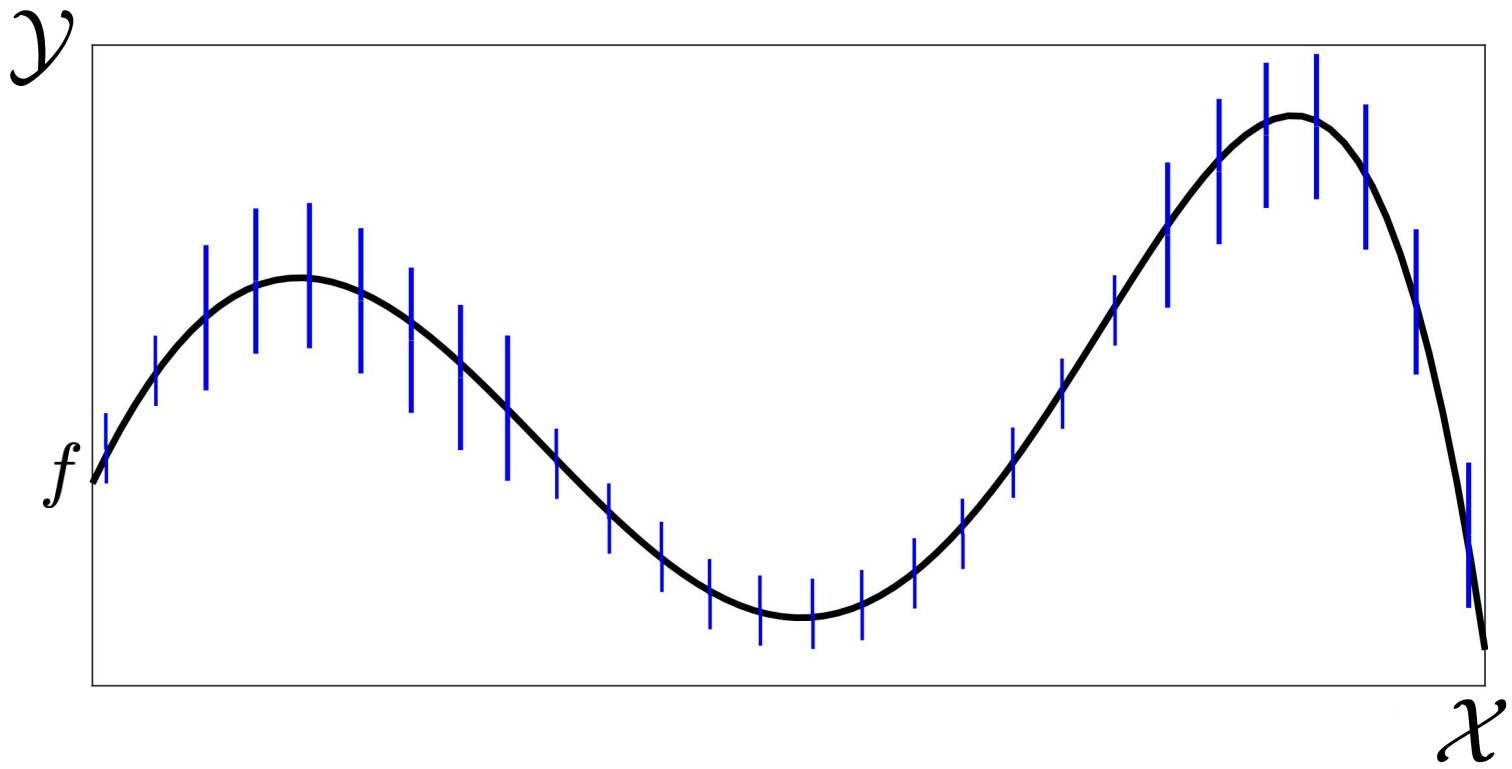
Often written

$$y \sim f(x) + \epsilon(x)$$

where:

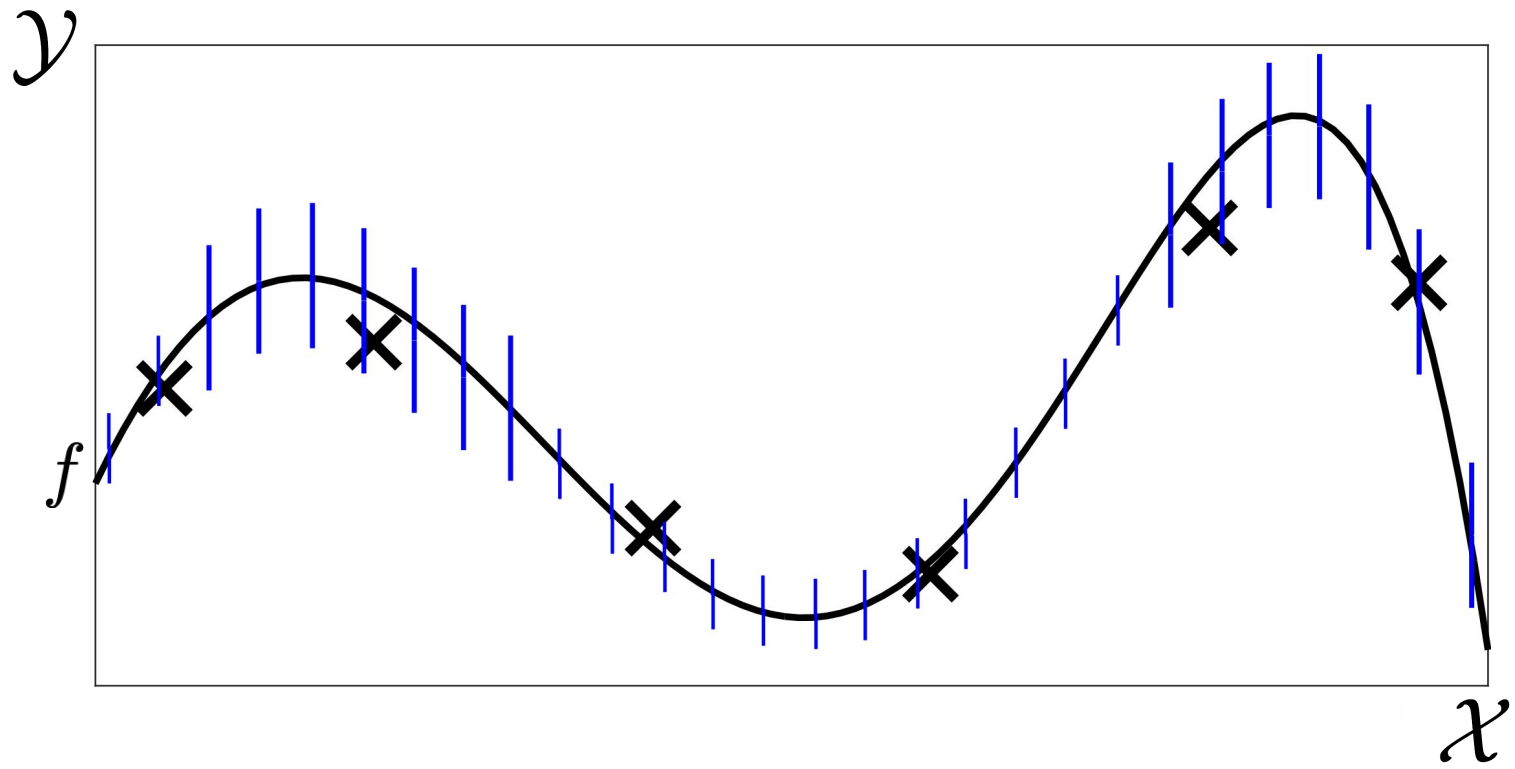
$f(x)$ Mean function (deterministic)

$\epsilon(x)$ Zero-mean noise (random variable).



Predictive UQ Models — Background and Typical Setup

Observations are sampled from this noisy function to yield the **dataset** $D_n = \left\{ \begin{bmatrix} x_1, y_1 \\ x_2, y_2 \\ \vdots \\ x_n, y_n \end{bmatrix} \right\}$

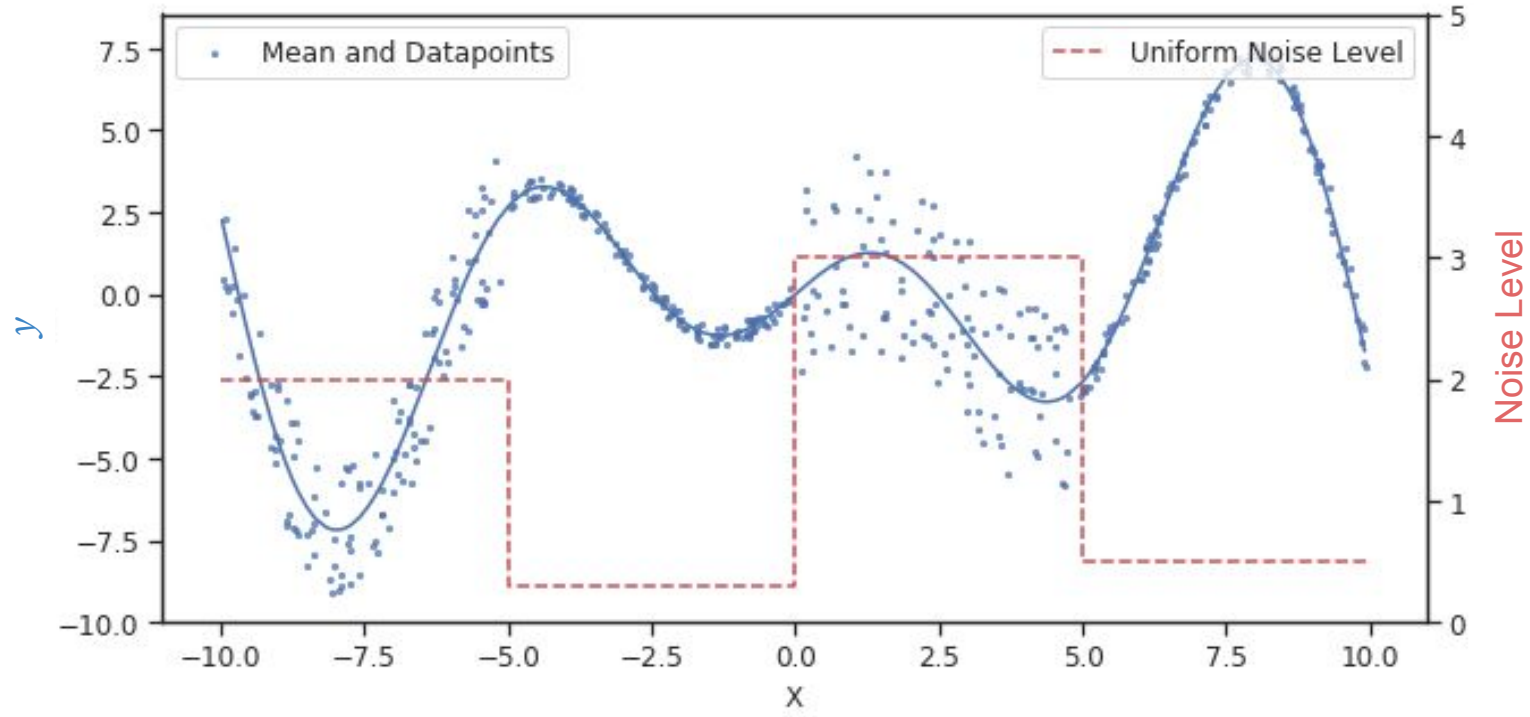


Predictive UQ Models — Background and Typical Setup

An aside:

Predictive UQ Models — Background and Typical Setup

An aside: noisy function may be **heteroscedastic**, *i.e.*, noise non-uniform wrt inputs x.



Source: Youngseog Chung

Predictive UQ Models — Background and Typical Setup

In practice, you only observe the dataset:

Predictive UQ Models — Background and Typical Setup

In practice, you only observe the dataset:

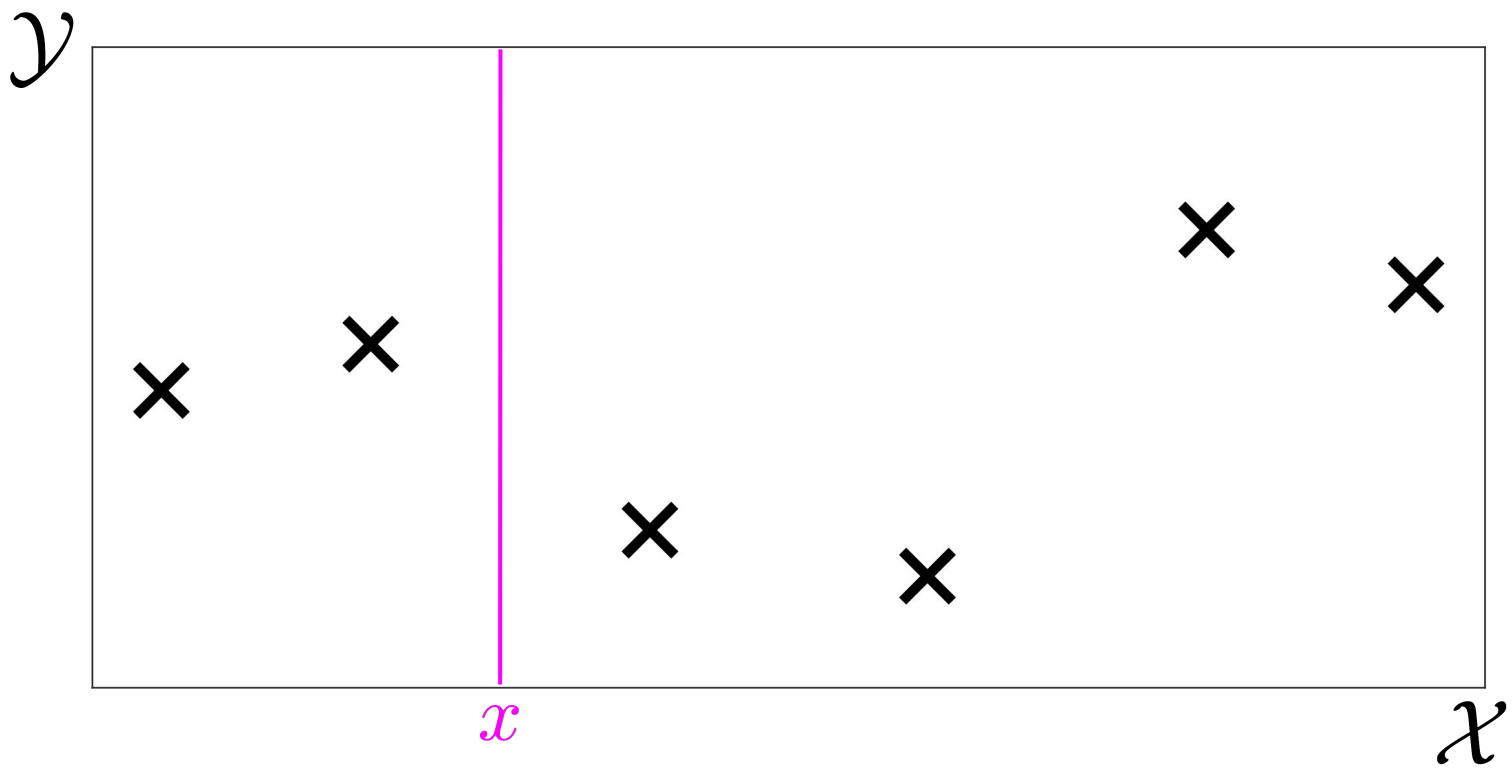
Predictive UQ Models — Background and Typical Setup

What are the goals of predictive UQ methods?

Predictive UQ Models — Background and Typical Setup

What are the goals of predictive UQ methods?

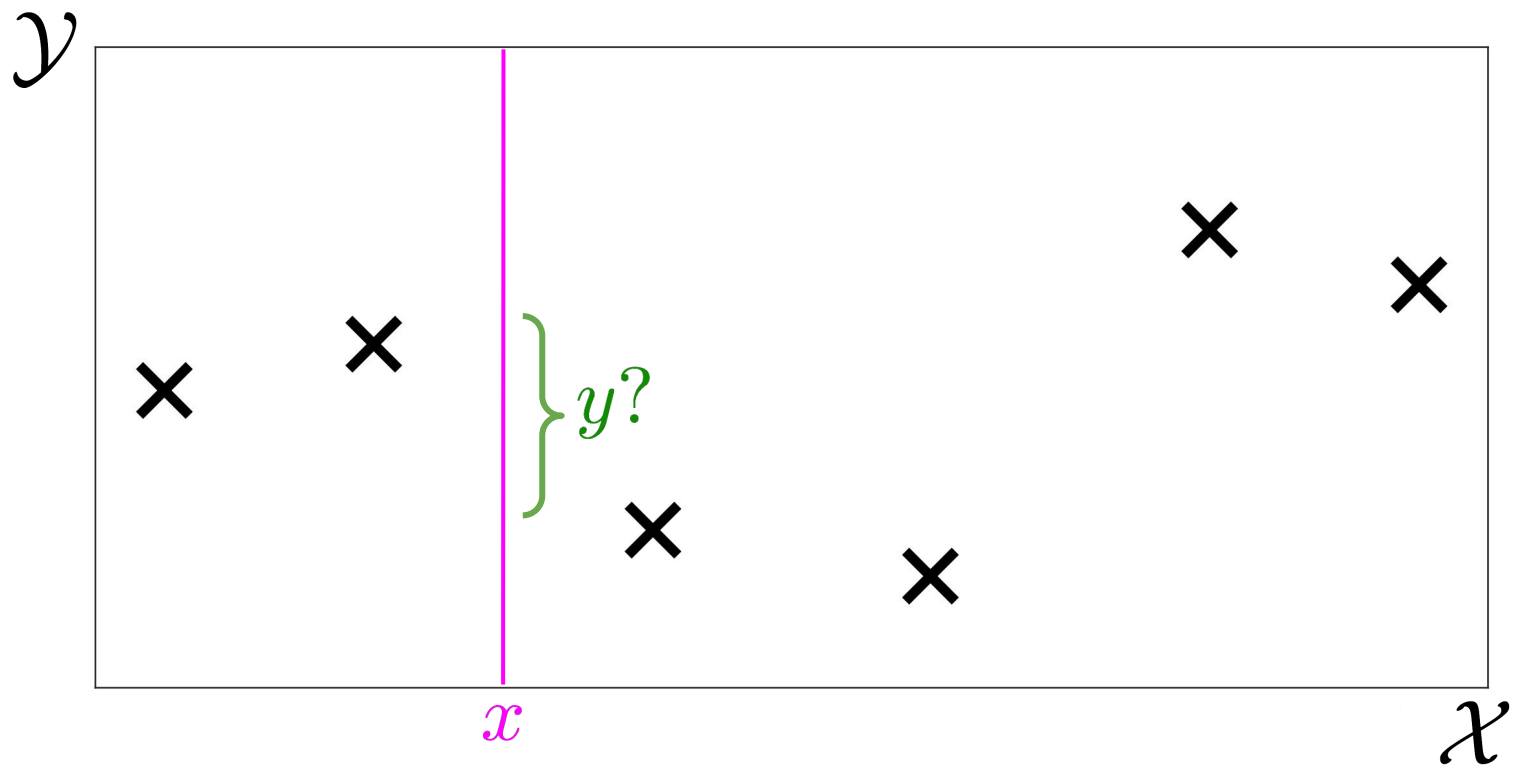
Intuitively, want to answer: for a given x,



Predictive UQ Models — Background and Typical Setup

What are the goals of predictive UQ methods?

Intuitively, want to answer: for a given x, what are the probable y values?

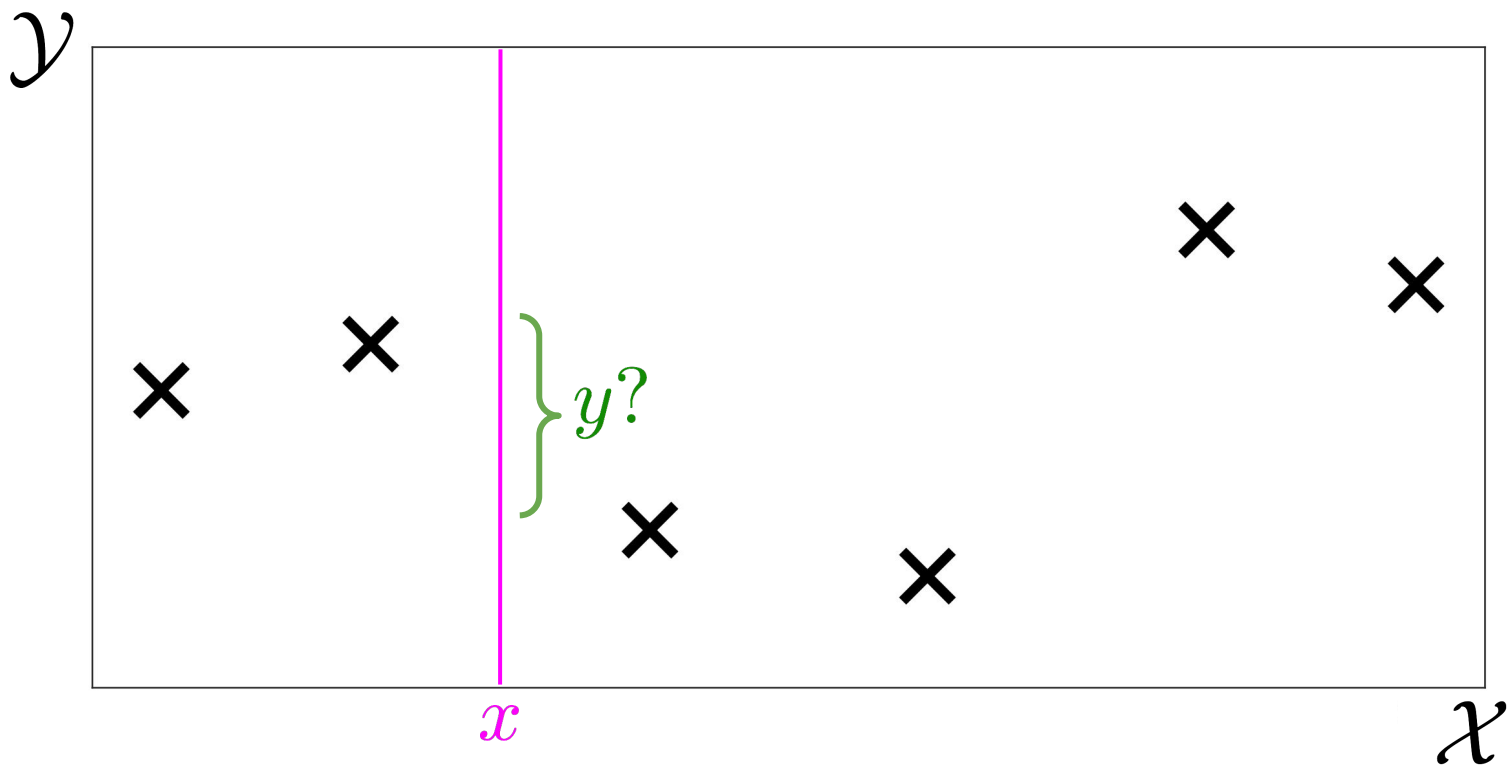


Predictive UQ Models — Background and Typical Setup

What are the goals of predictive UQ methods?

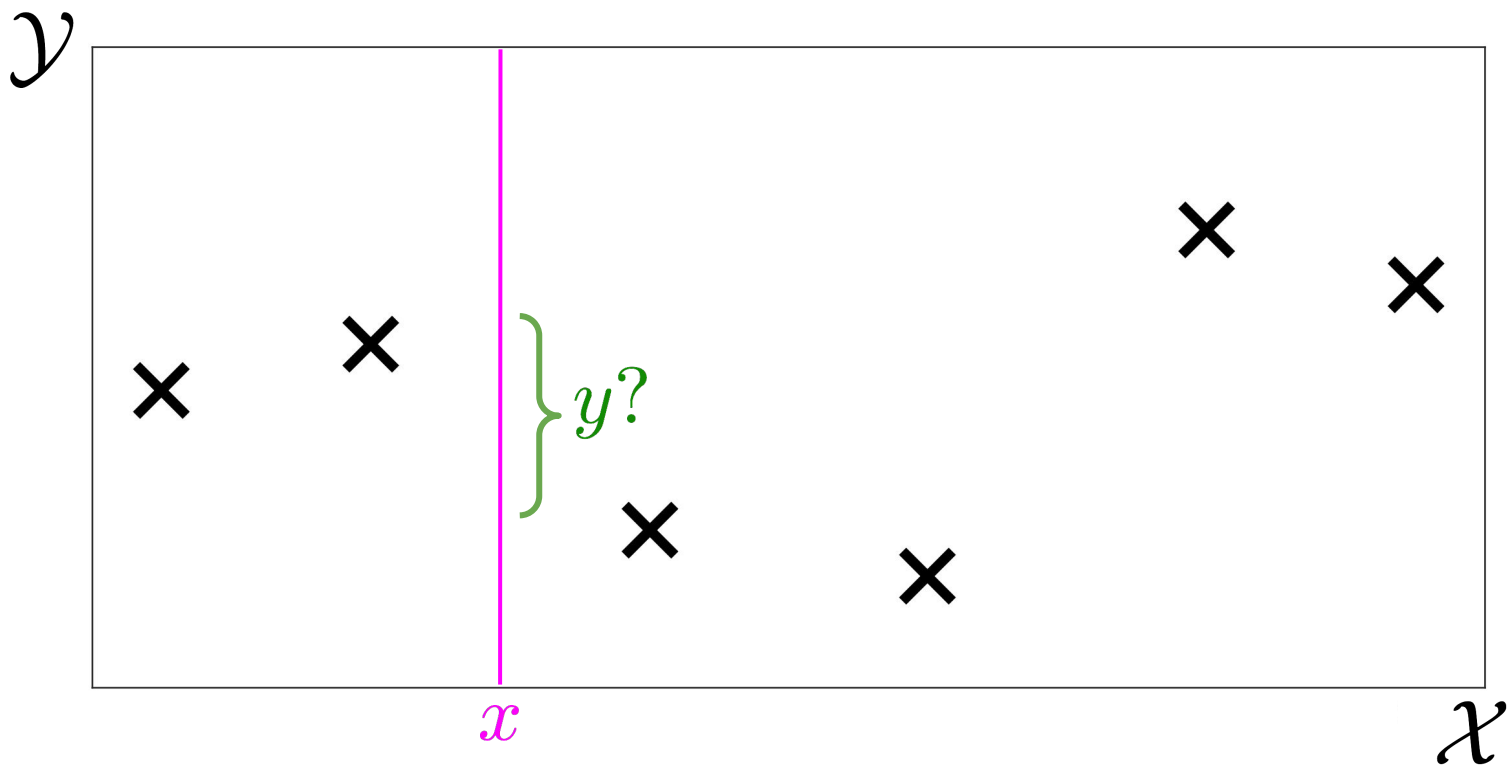
Intuitively, want to answer: for a given x, what are the probable y values?

Could return a distribution over y, or an interval over y, or samples, etc.



Predictive UQ Models — Background and Typical Setup

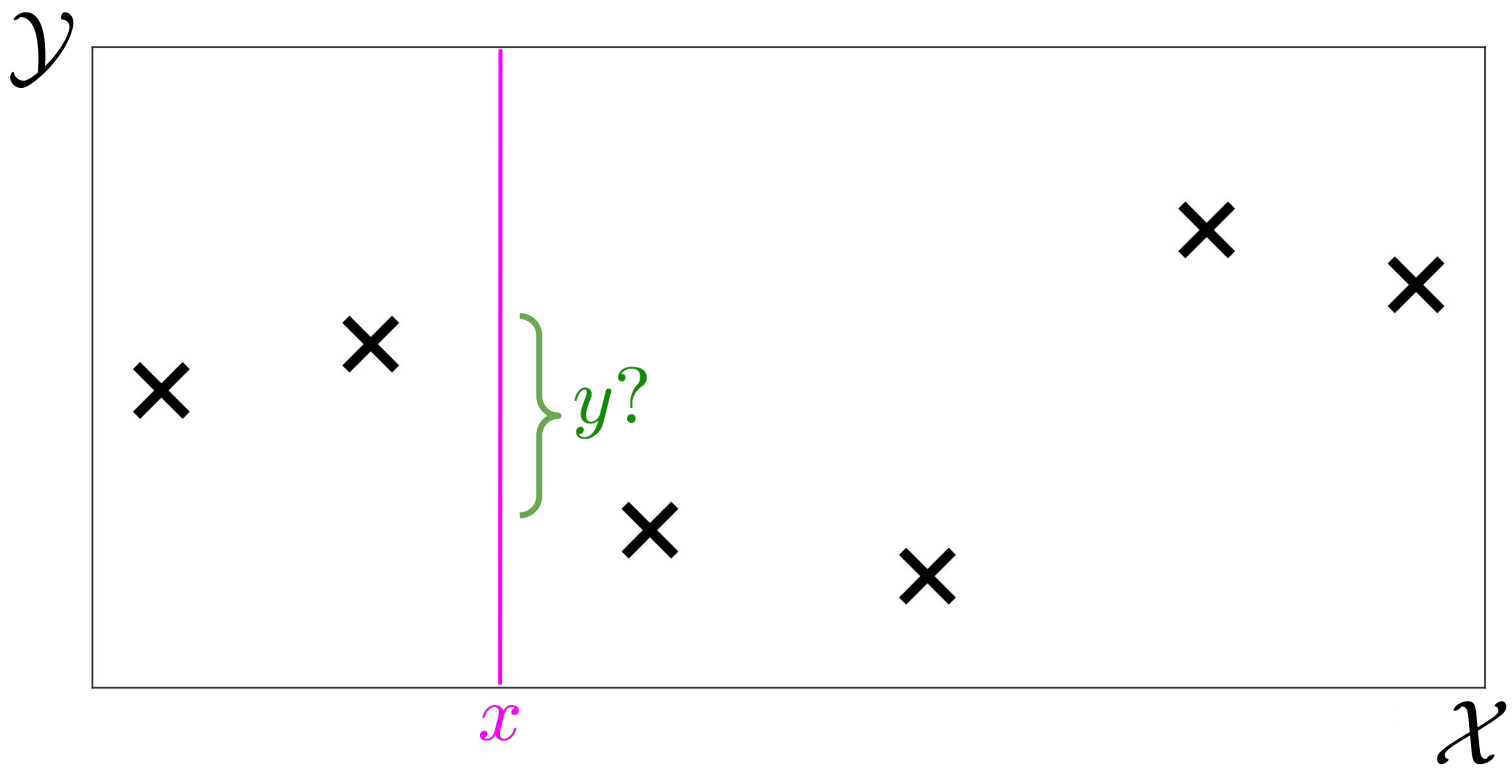
What are the *sources of uncertainty* that we might want to model? *(Why are we uncertain about y?)*



Predictive UQ Models — Background and Typical Setup

What are the *sources of uncertainty* that we might want to model? *(Why are we uncertain about y?)*

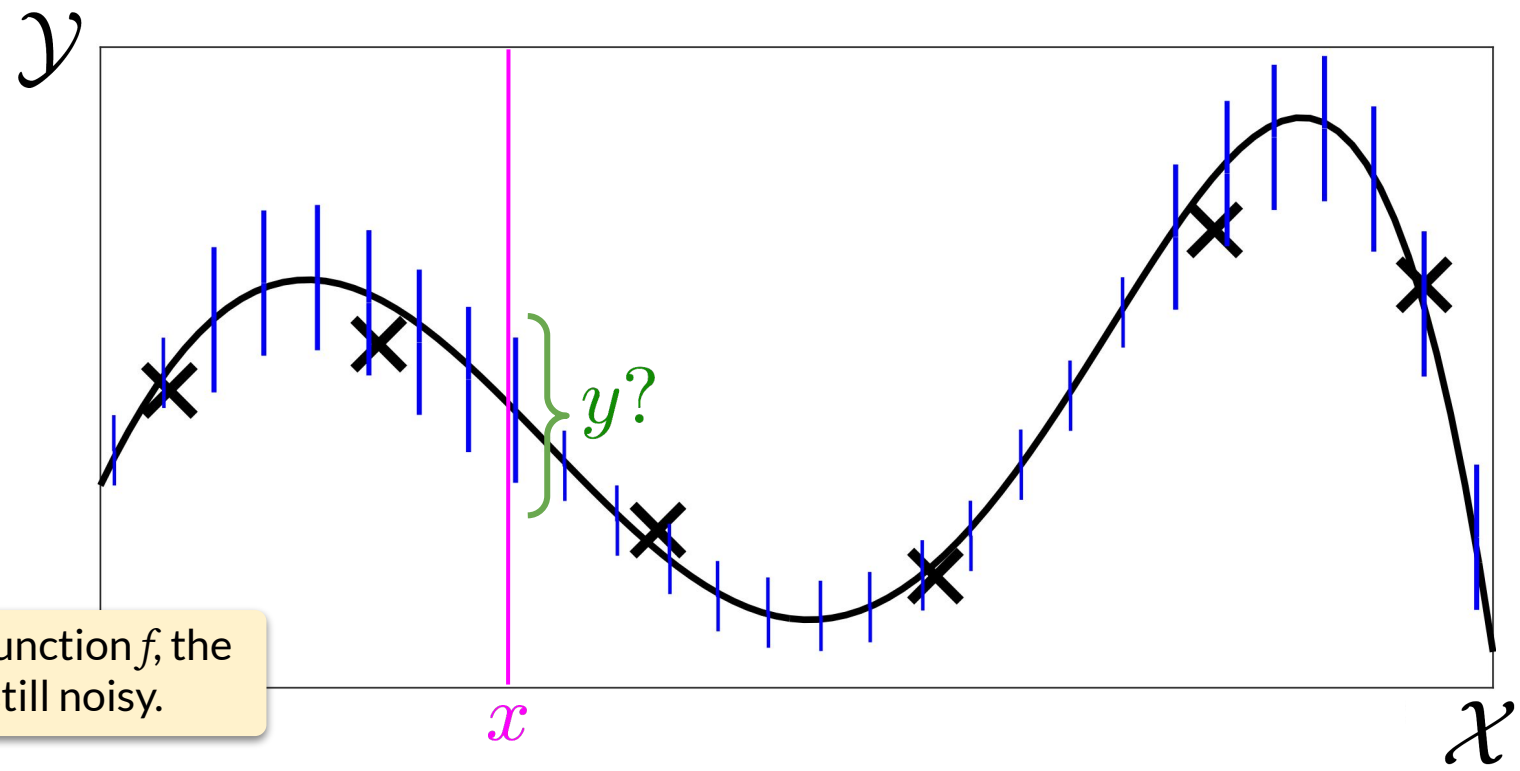
(1) Aleatoric uncertainty (irreducible uncertainty) — intrinsic noise in output, even with infinite observations.



Predictive UQ Models — Background and Typical Setup

What are the *sources of uncertainty* that we might want to model? *(Why are we uncertain about y?)*

(1) Aleatoric uncertainty (irreducible uncertainty) — intrinsic noise in output, even with infinite observations.

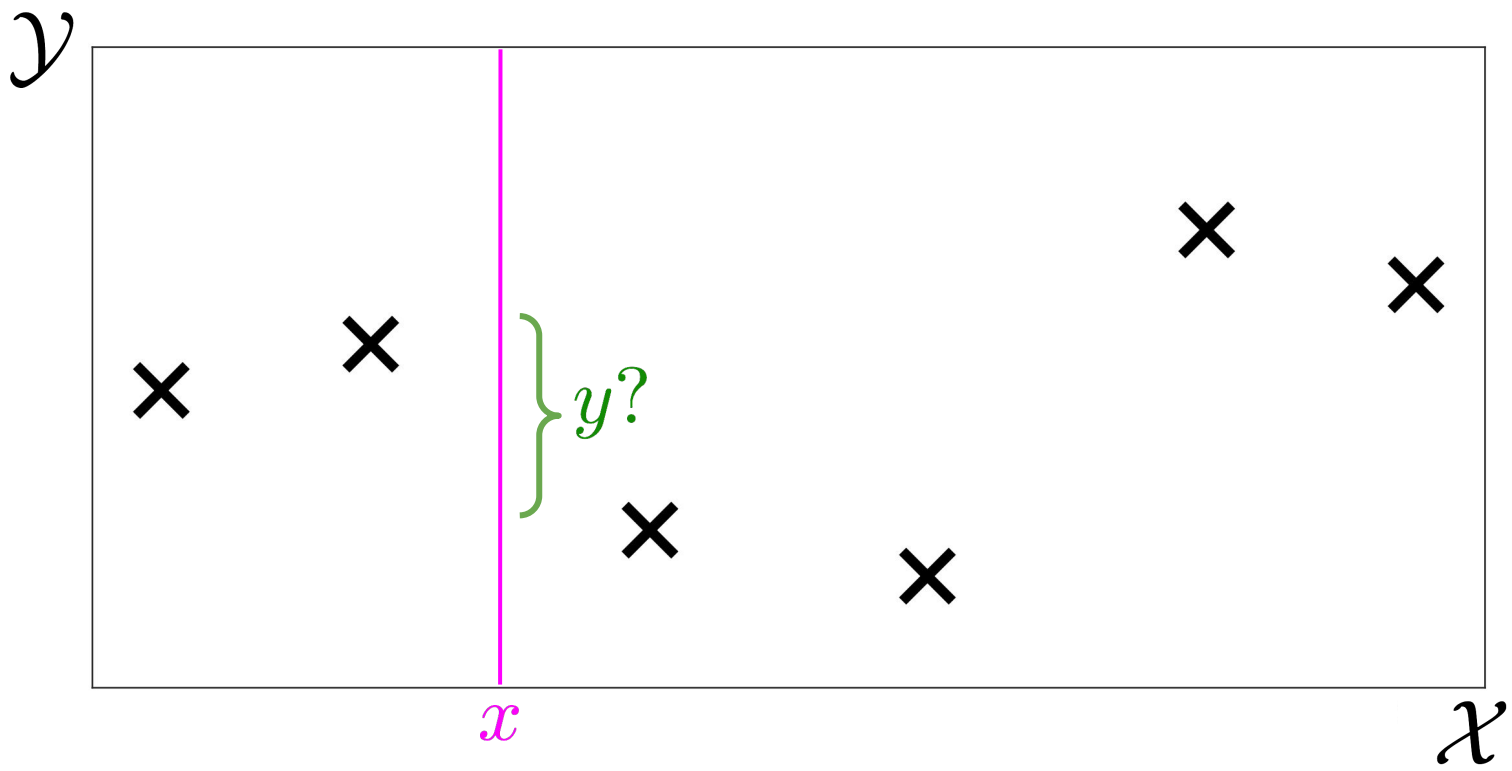


Even for a given, fixed function f, the observations y are still noisy.

Predictive UQ Models — Background and Typical Setup

What are the *sources of uncertainty* that we might want to model? *(Why are we uncertain about y?)*

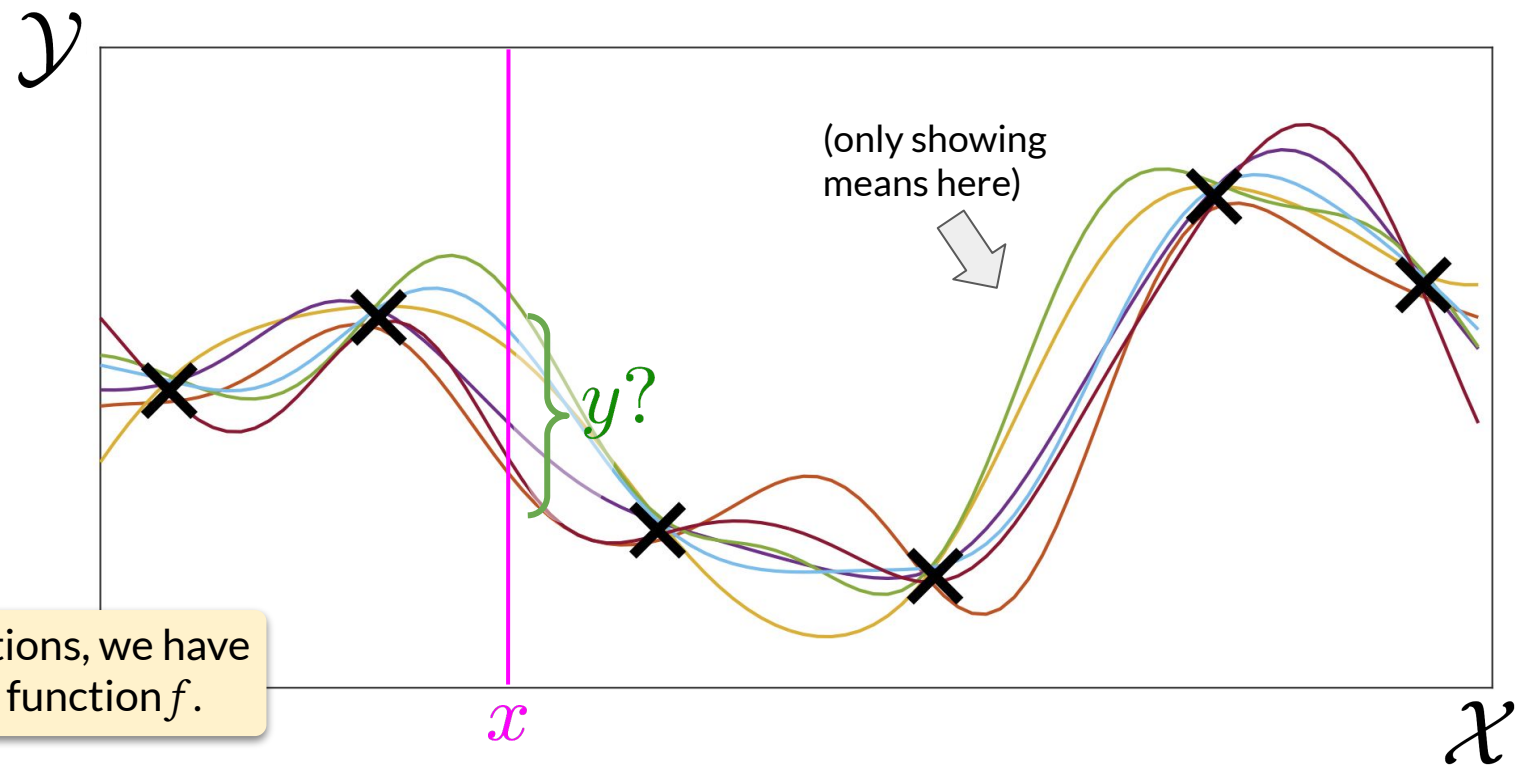
(1) Epistemic uncertainty — uncertainty due to lack of observations.



Predictive UQ Models — Background and Typical Setup

What are the *sources of uncertainty* that we might want to model? *(Why are we uncertain about y?)*

(1) Epistemic uncertainty — uncertainty due to lack of observations.



Due to a lack of observations, we have uncertainty about the function f.

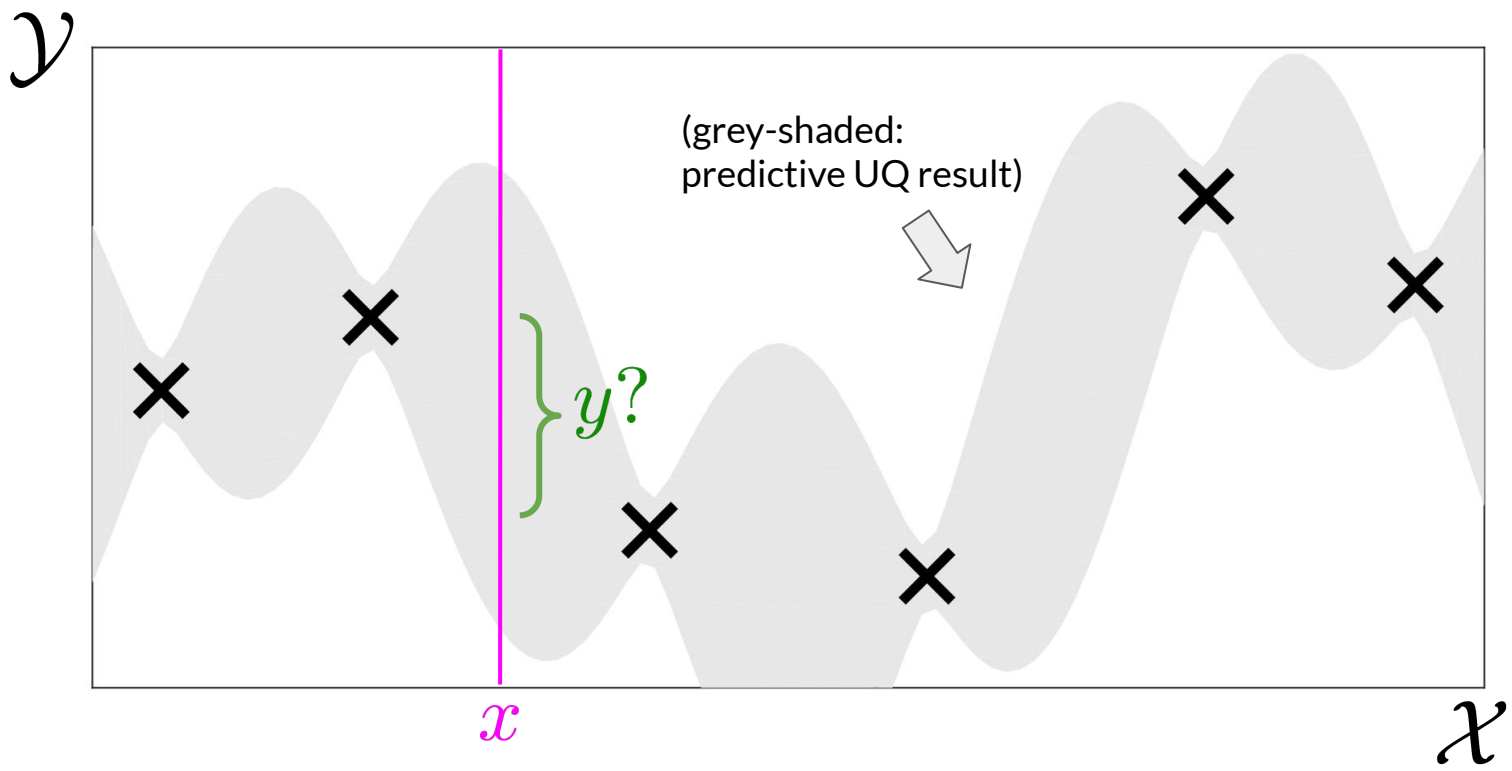
Predictive UQ Models — Background and Typical Setup

So what do predictive UQ methods compute/return?

Predictive UQ Models — Background and Typical Setup

So what do predictive UQ methods compute/return?

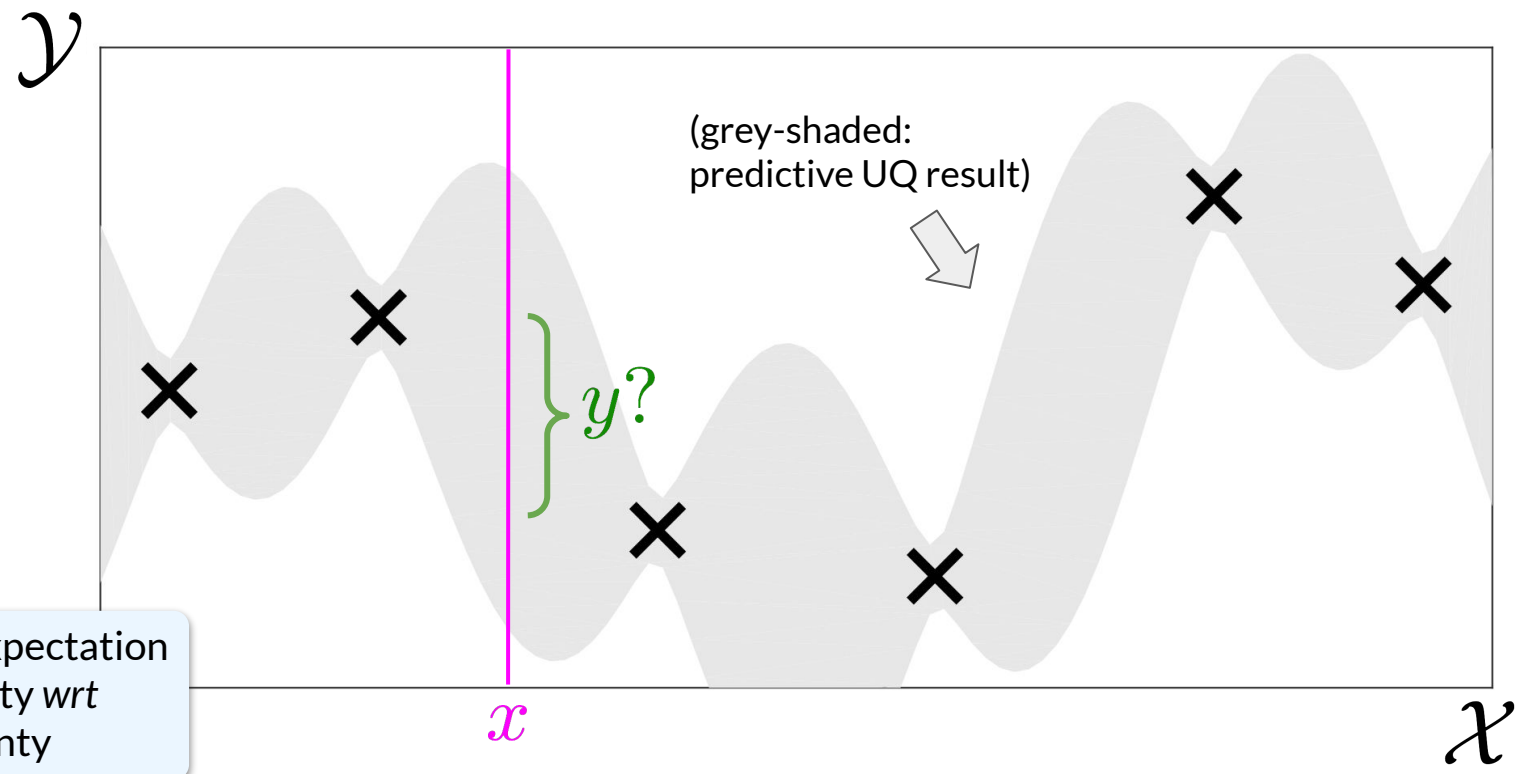
Often, predictive UQ methods try to return some sort of **combination** of aleatoric and epistemic uncertainty.



Predictive UQ Models — Background and Typical Setup

So what do predictive UQ methods compute/return?

Often, predictive UQ methods try to return some sort of **combination** of aleatoric and epistemic uncertainty.



E.g., marginalize — take expectation of aleatoric uncertainty *wrt* epistemic uncertainty

Predictive UQ Models — Background and Typical Setup

Example: Bayesian linear model with homoscedastic Gaussian noise.

Predictive UQ Models — Background and Typical Setup

Example: Bayesian linear model with homoscedastic Gaussian noise.

Prior: $\theta \sim p(\theta),$ Likelihood: $y \sim p(y \mid \theta\ ;\ x) = \mathcal{N}(\theta^\top x, \sigma^2)$

Predictive UQ Models — Background and Typical Setup

Example: Bayesian linear model with homoscedastic Gaussian noise.

Prior: $\theta \sim p(\theta)$, Likelihood: $y \sim p(y \mid \theta\ ;\ x) = \mathcal{N}(\theta^\top x, \sigma^2)$

Epistemic uncertainty over linear model parameters
(here: posterior distribution)

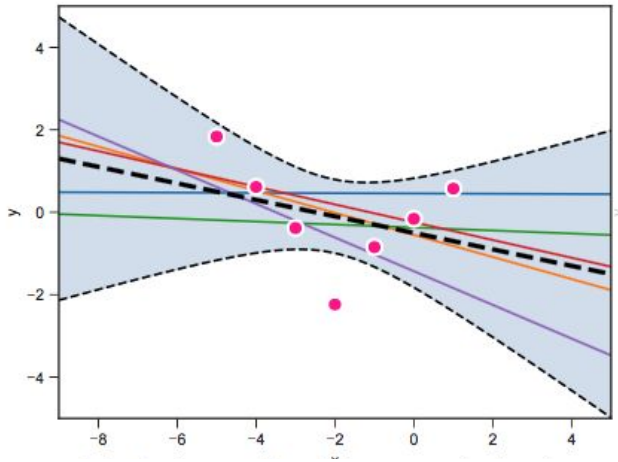
Aleatoric uncertainty shown for a single set of model parameters
(here: for MAP parameters)

Example predictive UQ result:
here we show the posterior predictive distribution.

Predictive UQ Models — Background and Typical Setup

Example: Bayesian linear model with homoscedastic Gaussian noise.

Prior: $\theta \sim p(\theta)$, Likelihood: $y \sim p(y \mid \theta\ ;\ x) = \mathcal{N}(\theta^\top x, \sigma^2)$



Epistemic uncertainty over linear model parameters
(here: posterior distribution)

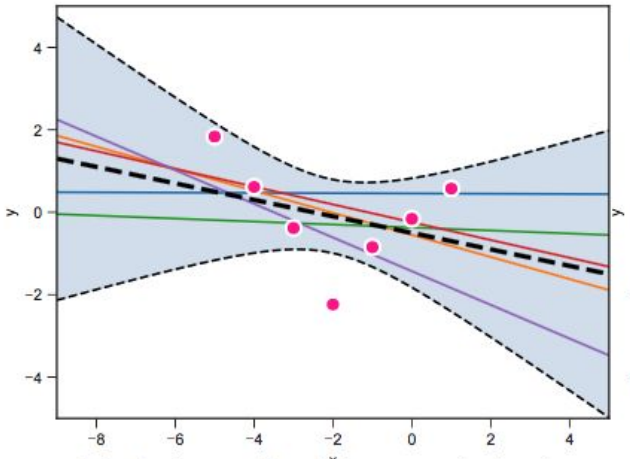
Aleatoric uncertainty shown for a single set of model parameters
(here: for MAP parameters)

Example predictive UQ result:
here we show the posterior predictive distribution.

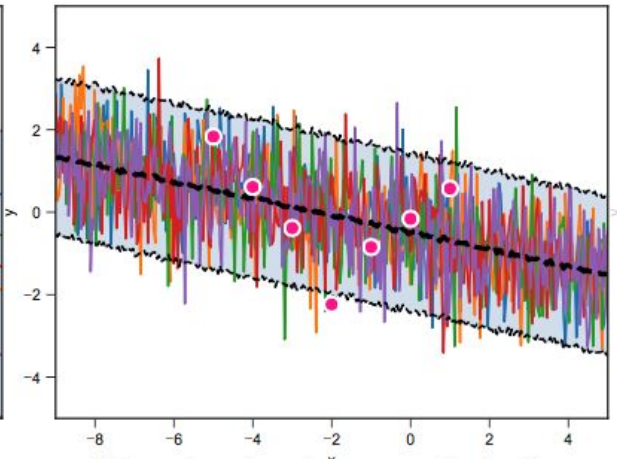
Predictive UQ Models — Background and Typical Setup

Example: Bayesian linear model with homoscedastic Gaussian noise.

Prior: $\theta \sim p(\theta)$, Likelihood: $y \sim p(y \mid \theta\ ;\ x) = \mathcal{N}(\theta^\top x, \sigma^2)$



Epistemic uncertainty over linear model parameters
(here: posterior distribution)



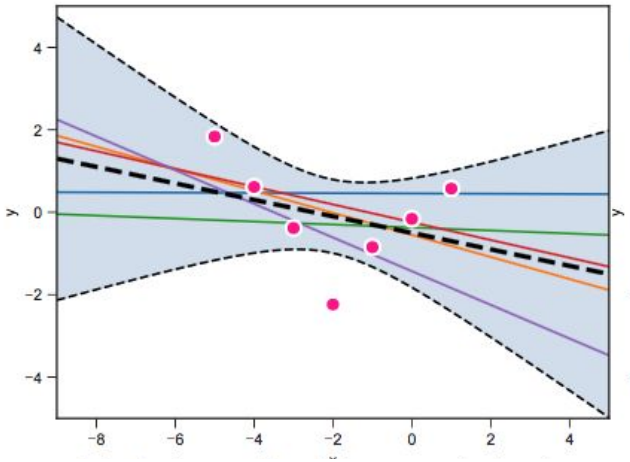
Aleatoric uncertainty shown for a single set of model parameters
(here: for MAP parameters)

Example predictive UQ result:
here we show the posterior predictive distribution.

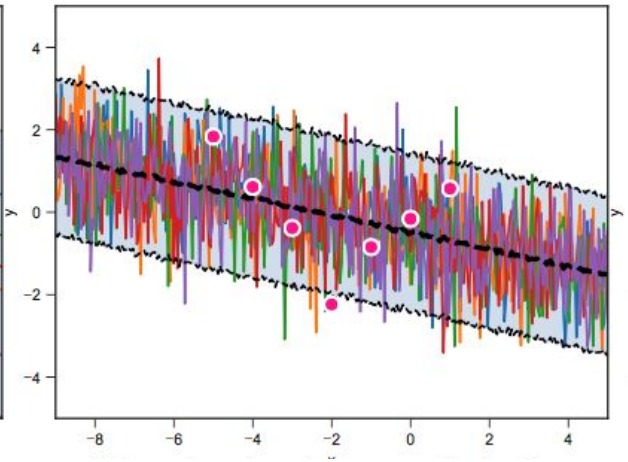
Predictive UQ Models — Background and Typical Setup

Example: Bayesian linear model with homoscedastic Gaussian noise.

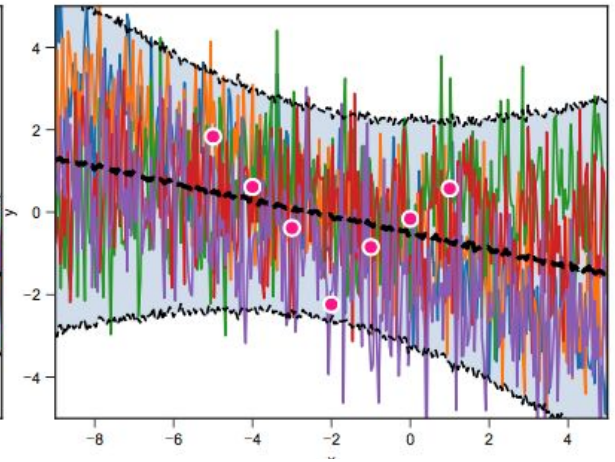
Prior: $\theta \sim p(\theta)$, Likelihood: $y \sim p(y \mid \theta\ ;\ x) = \mathcal{N}(\theta^\top x, \sigma^2)$



Epistemic uncertainty over linear model parameters
(here: posterior distribution)



Aleatoric uncertainty shown for a single set of model parameters
(here: for MAP parameters)



Example predictive UQ result:
here we show the posterior predictive distribution.

A Note on PGMs for Predictive UQ Models

A quick note on the PGM structure for predictive UQ models.

A Note on PGMs for Predictive UQ Models

A quick note on the PGM structure for predictive UQ models.

Suppose that we have a (probabilistic) predictive UQ model consisting of:

- Inputs x — input features
- Outputs y — output labels (assuming continuous / regression setting)
- Parameters θ — defining model from some family of models.

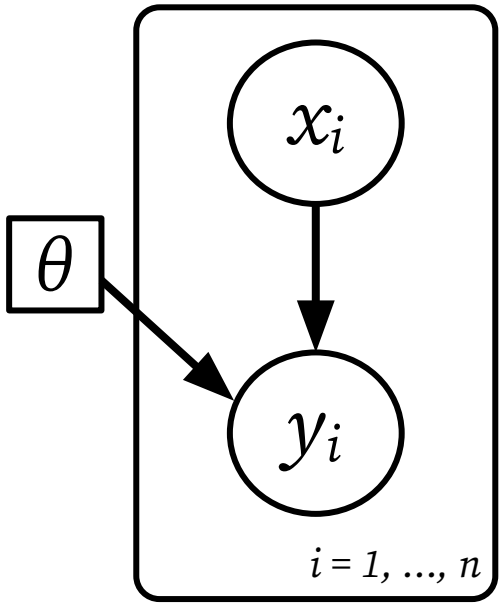
A Note on PGMs for Predictive UQ Models

Based on previous lectures in this class, you might expect a PGM structure that looks something like the following:

A Note on PGMs for Predictive UQ Models

Based on previous lectures in this class, you might expect a PGM structure that looks something like the following:

Joint PDF over Inputs/Outputs



Generative Process

Prior: $p(x_i)$

Likelihood: $p_\theta(y_i \mid x_i)$

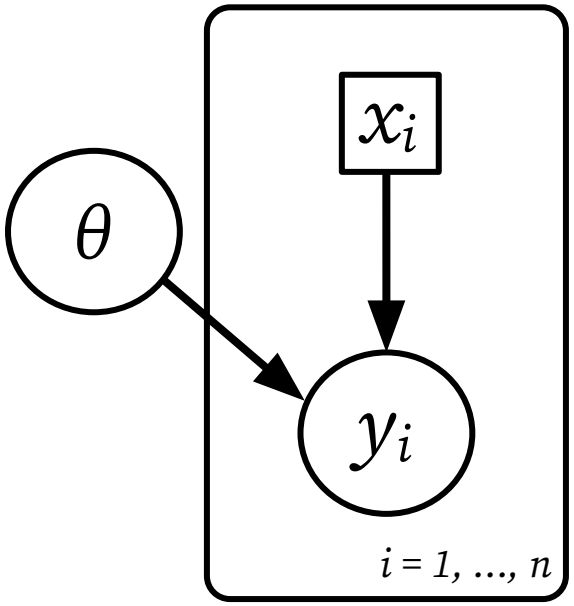
A Note on PGMs for Predictive UQ Models

But instead, the PGMs for predictive UQ models (*probabilistic supervised ML models*) often look like this:

A Note on PGMs for Predictive UQ Models

But instead, the PGMs for predictive UQ models (*probabilistic supervised ML models*) often look like this:

Prior on Parameter



Generative Process

Prior: $p(\theta)$

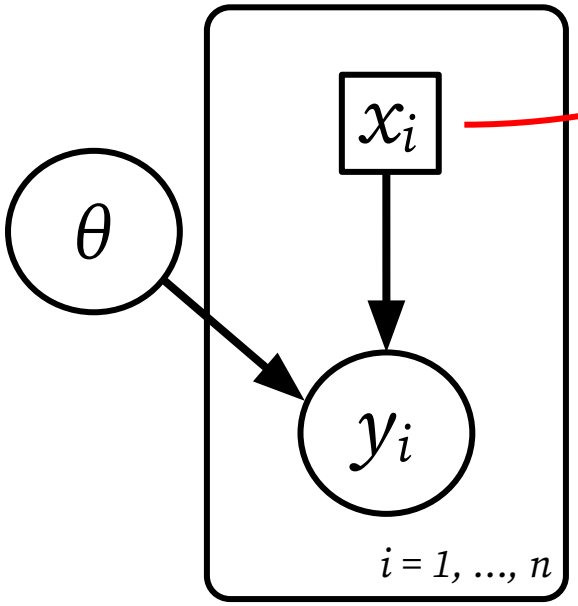
Likelihood: $p(y_i \mid \theta \; ; \; x_i)$

A Note on PGMs for Predictive UQ Models

But instead, the PGMs for predictive UQ models (*probabilistic supervised ML models*) often look like this:

Key detail: inputs x are considered parameters (not random variables)

Prior on Parameter



Generative Process

Prior: $p(\theta)$

Likelihood: $p(y_i \mid \theta \; ; \; x_i)$

A Note on PGMs for Predictive UQ Models

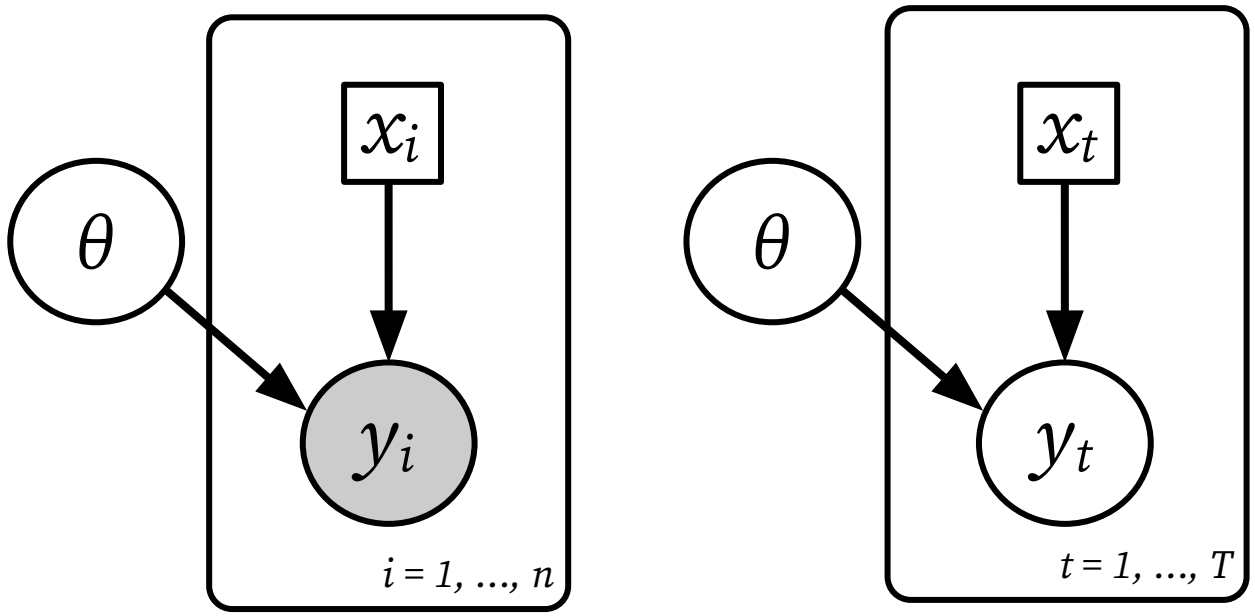
But instead, the PGMs for predictive UQ models (*probabilistic supervised ML models*) often look like this:

(Or like this:)

A Note on PGMs for Predictive UQ Models

But instead, the PGMs for predictive UQ models (*probabilistic supervised ML models*) often look like this:

(Or like this:)



Train

Test

Next Steps

In rest of lecture we will:

Next Steps

In rest of lecture we will:

- Go through a few popular *predictive UQ models*.
- Starting with **Gaussian Processes**.
- Then describing some **Deep Uncertainty Models** (neural network-based).
  - Including: some neural network + GP hybrid models.

Gaussian Processes

Gaussian Processes

A very popular type of predictive UQ model is called a **Gaussian Process (GP).**

Gaussian Processes

Also referred to as a *Bayesian* or *nonparametric Bayesian* model

A very popular type of predictive UQ model is called a **Gaussian Process (GP).**

Gaussian Processes

Also referred to as a *Bayesian* or *nonparametric Bayesian* model

A very popular type of predictive UQ model is called a **Gaussian Process (GP).**

This model is used heavily for tasks like *Bayesian optimization*, *active learning*, even some probabilistic *reinforcement learning*.

Also used in the sciences, including physics, biology, and geostatistics
(for tasks like optimization, simulation, and modeling physical systems)

Gaussian Processes

Also referred to as a *Bayesian* or *nonparametric Bayesian* model

A very popular type of predictive UQ model is called a **Gaussian Process (GP).**

This model is used heavily for tasks like *Bayesian optimization*, *active learning*, even some probabilistic *reinforcement learning*.

Also used in the sciences, including physics, biology, and geostatistics
(for tasks like optimization, simulation, and modeling physical systems)

I aim to give a basic overview of GPs (a full overview could take a full lecture or more).

⇒ Later, I'll show some examples of models that combine GPs and neural networks.

Gaussian Processes — Intuition

Before getting into the math, I want to give a high-level intuition…

Gaussian Processes — Intuition

Before getting into the math, I want to give a high-level intuition…

A *GP* is a probabilistic model, which can be viewed as a distribution over functions.

But primary uses (multivariate) **Gaussian distribution** to model these functions 🤯!

Gaussian Processes — Intuition

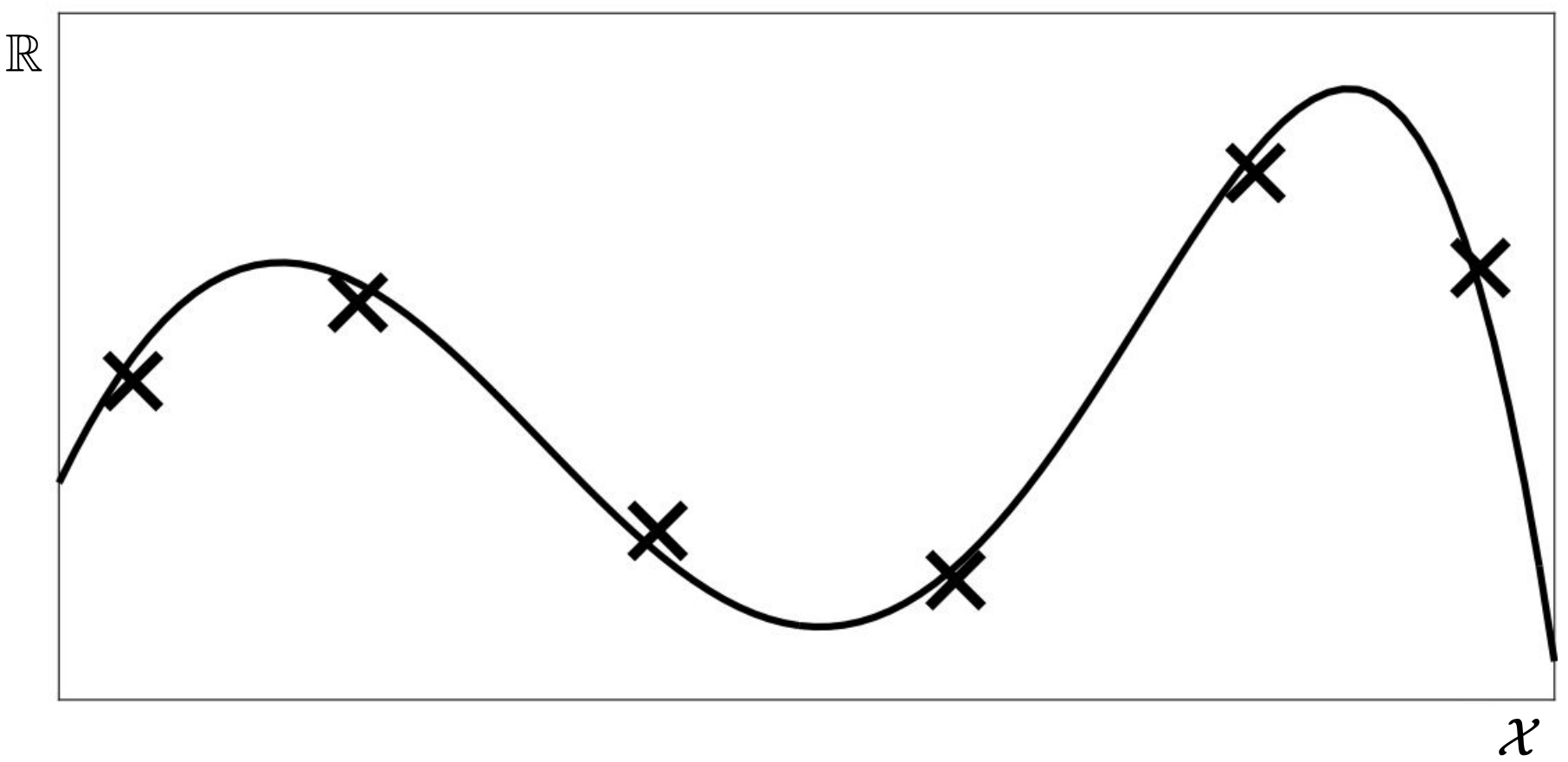
Before getting into the math, I want to give a high-level intuition…

A *GP* is a probabilistic model, which can be viewed as a distribution over functions.

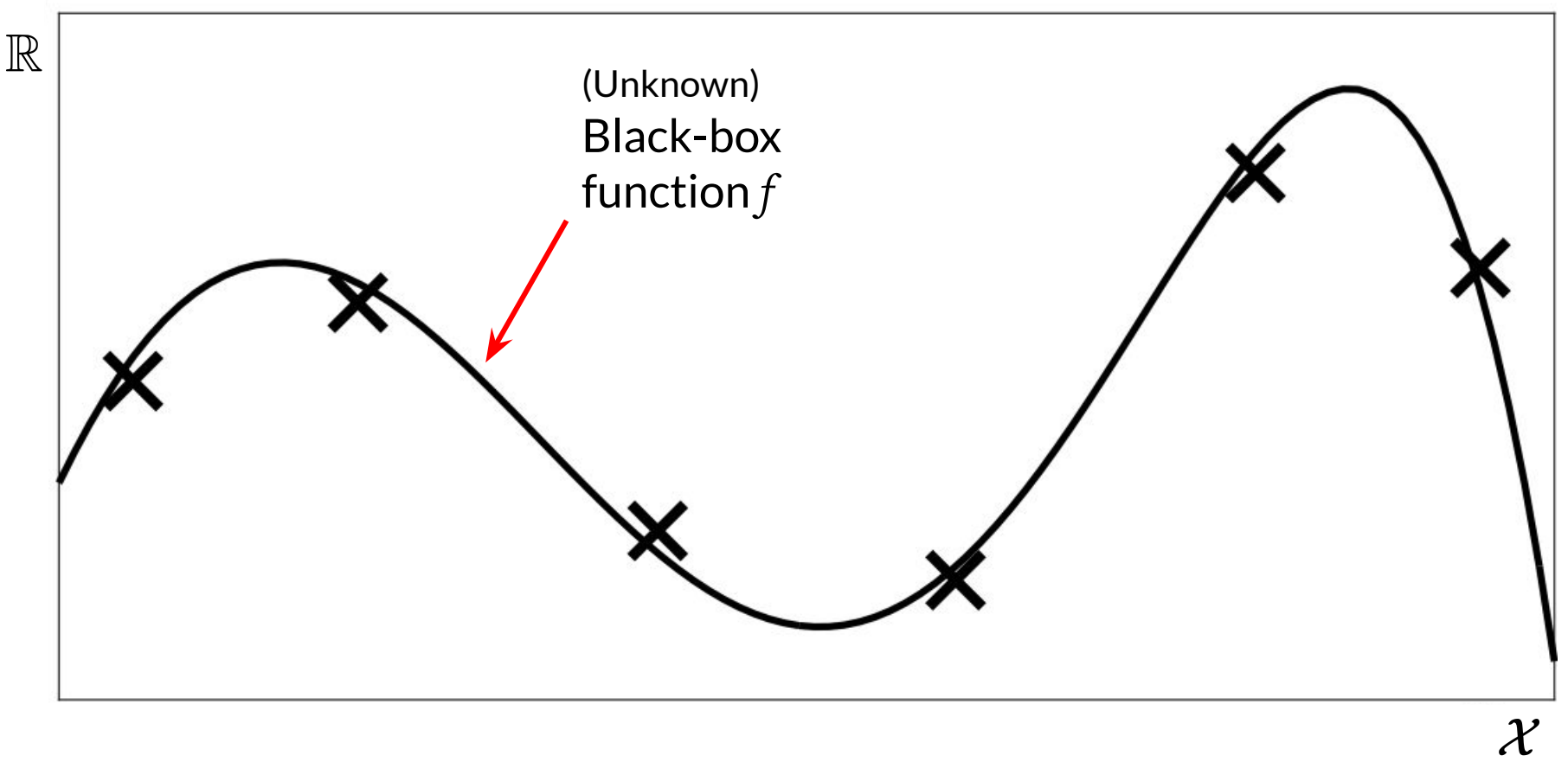
But primary uses (multivariate) **Gaussian distribution** to model these functions 🤯!

Lets visualize this…

Gaussian Processes — Intuition

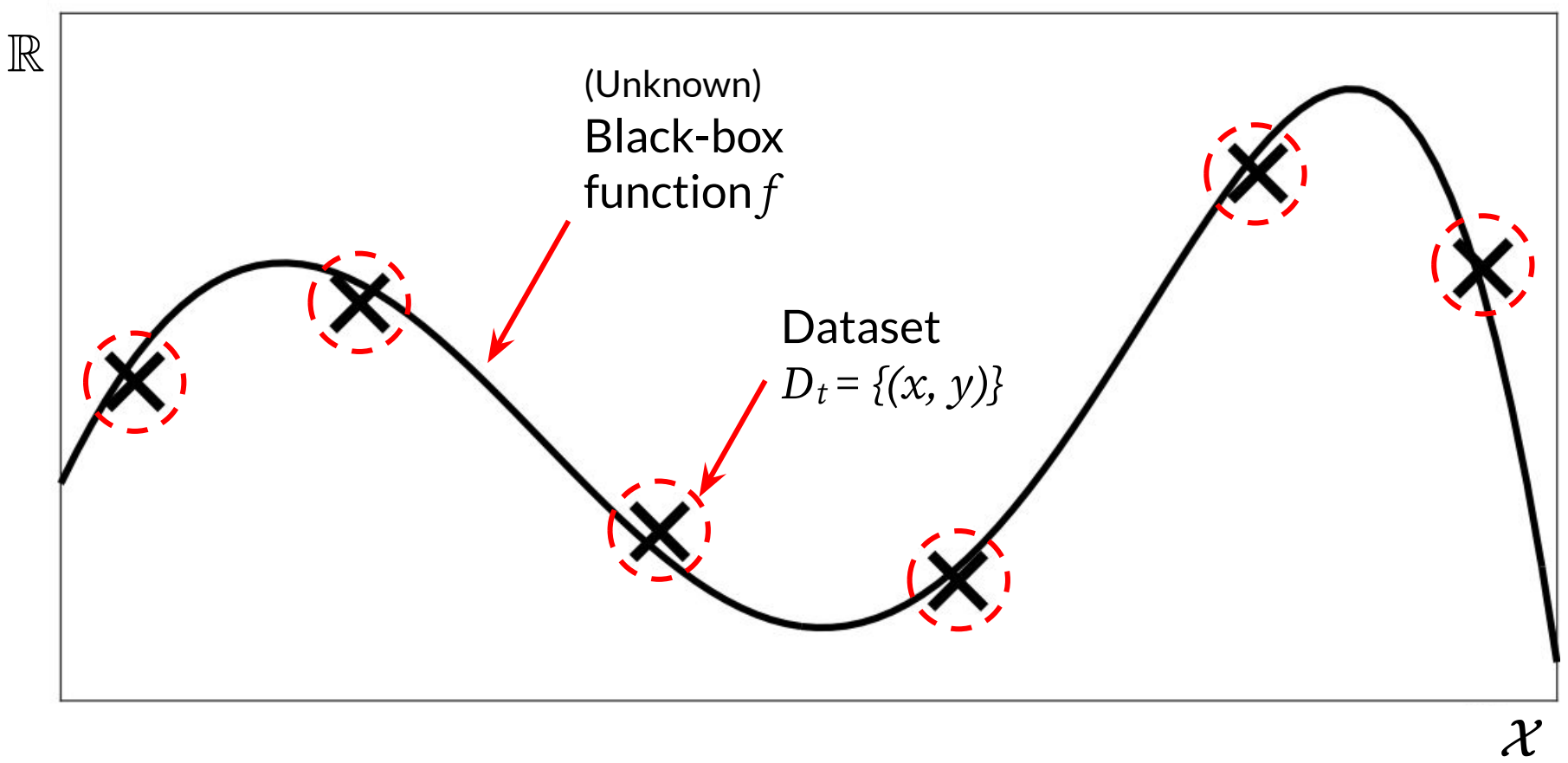


Gaussian Processes — Intuition



Unknown black-box f

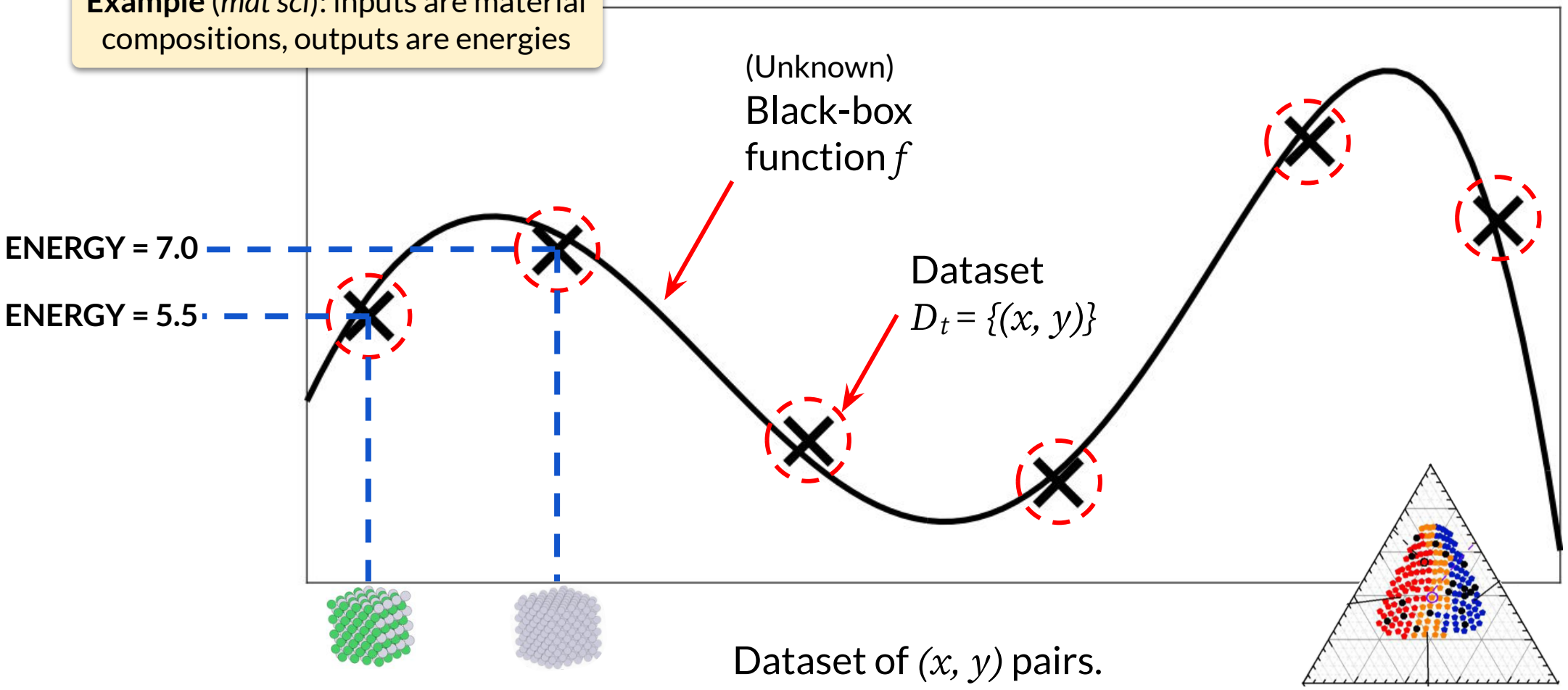
Gaussian Processes — Intuition



Unknown black-box f, and dataset of (x, y) pairs.

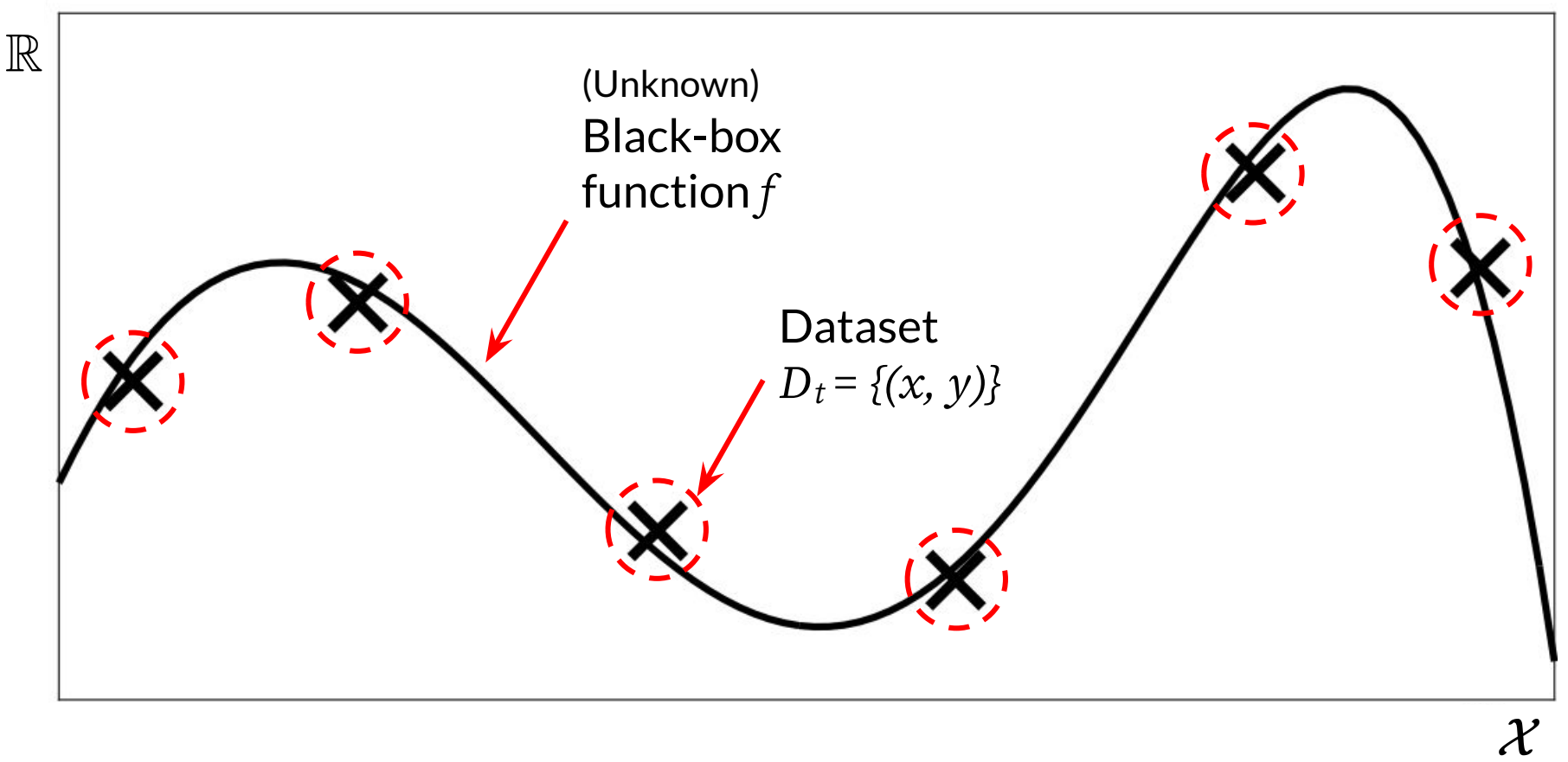
Gaussian Processes — Intuition

Example (*mat sci*): inputs are material compositions, outputs are energies



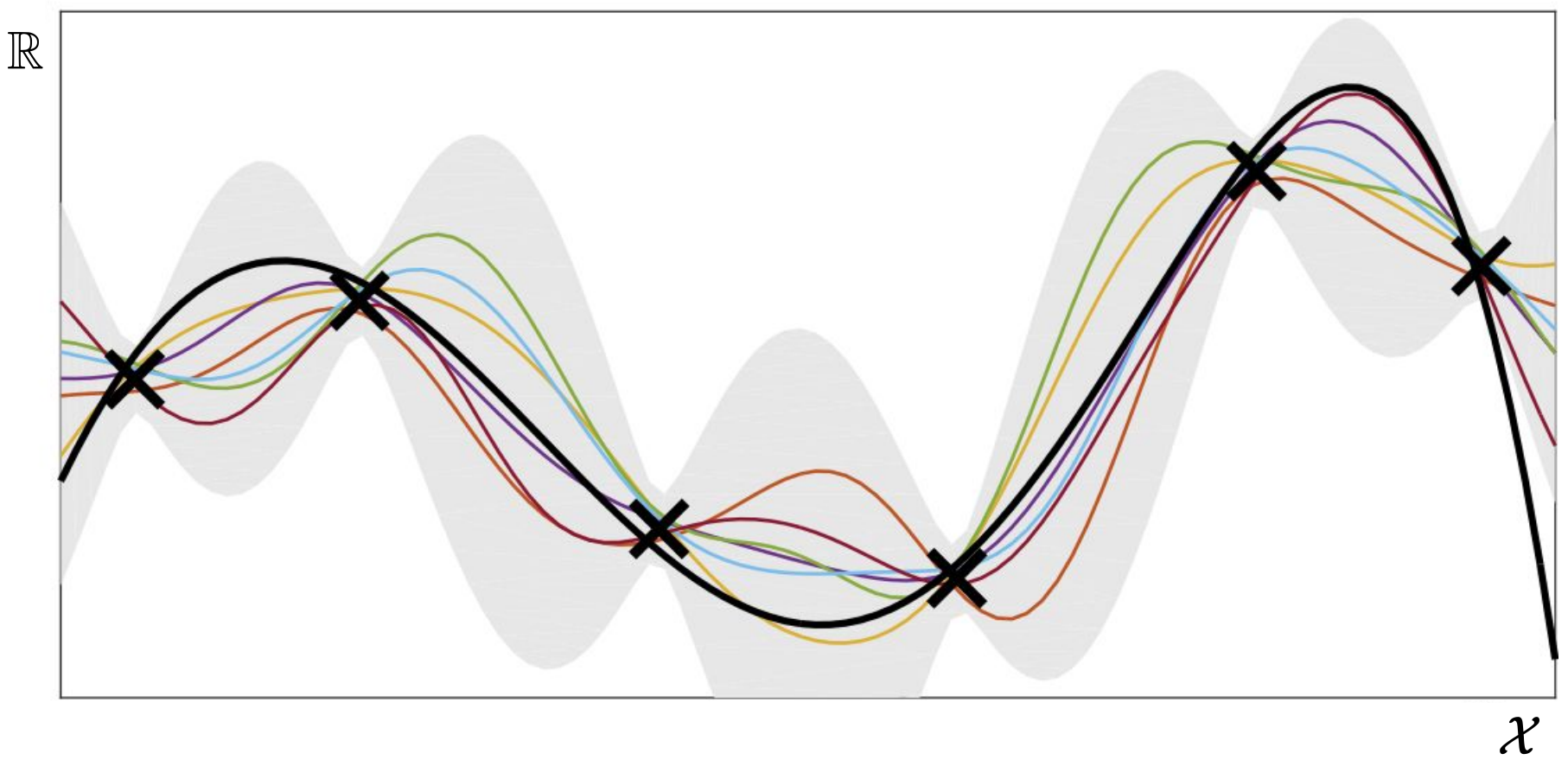
Dataset of (x, y) pairs.

Gaussian Processes — Intuition



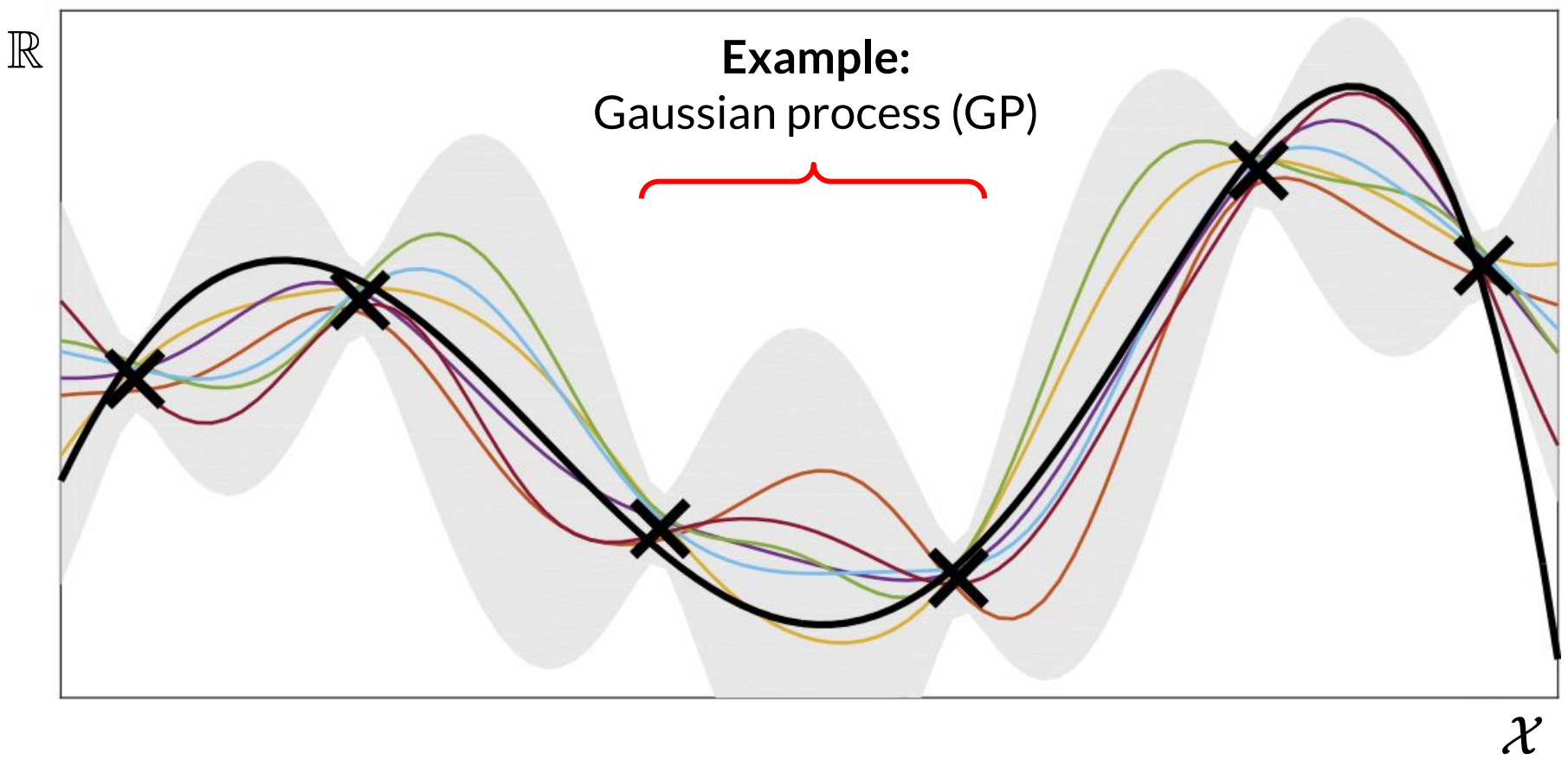
Dataset of (x, y) pairs.

Gaussian Processes — Intuition



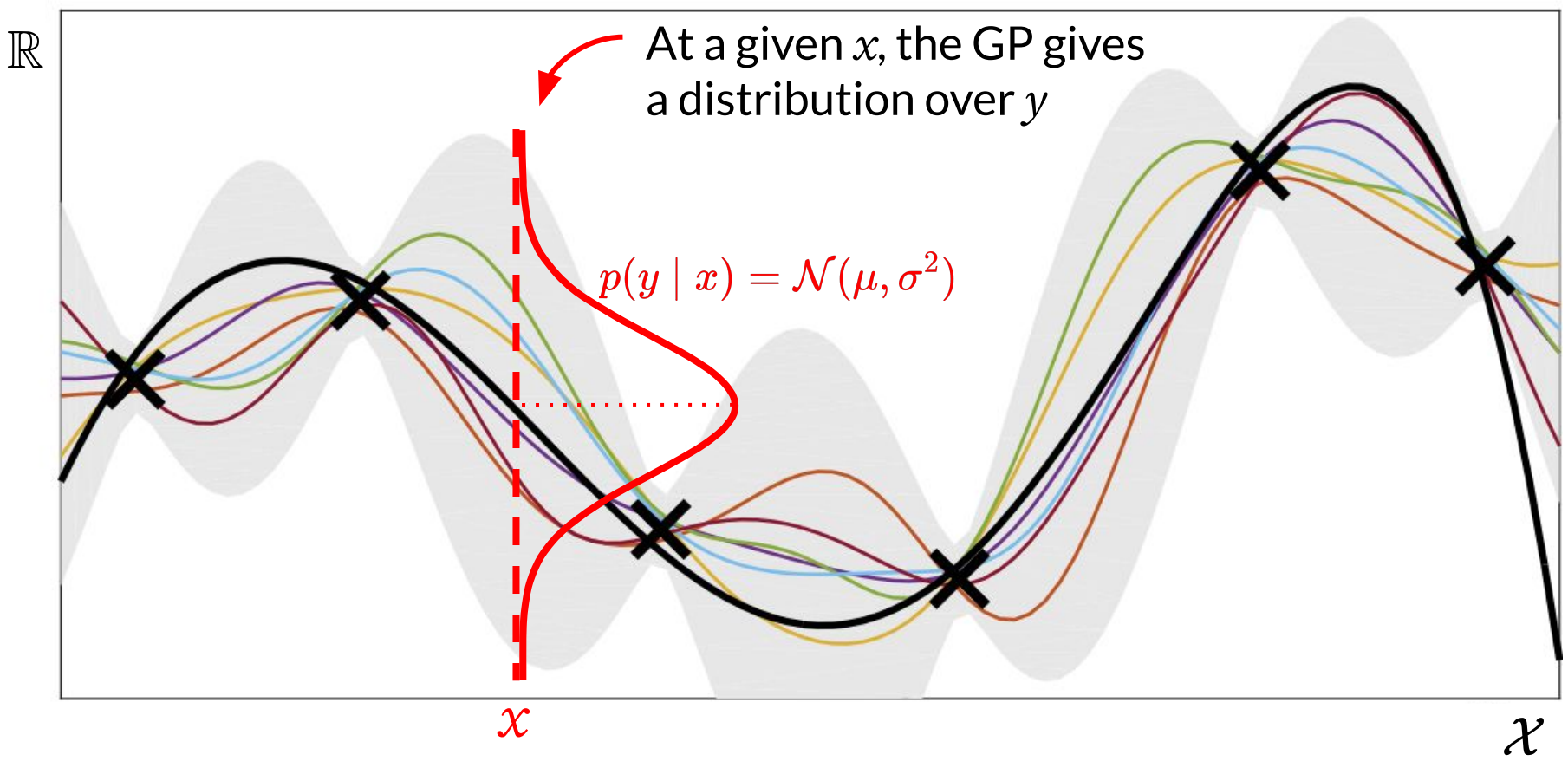
Given this dataset, can fit *predictive uncertainty* model.

Gaussian Processes — Intuition



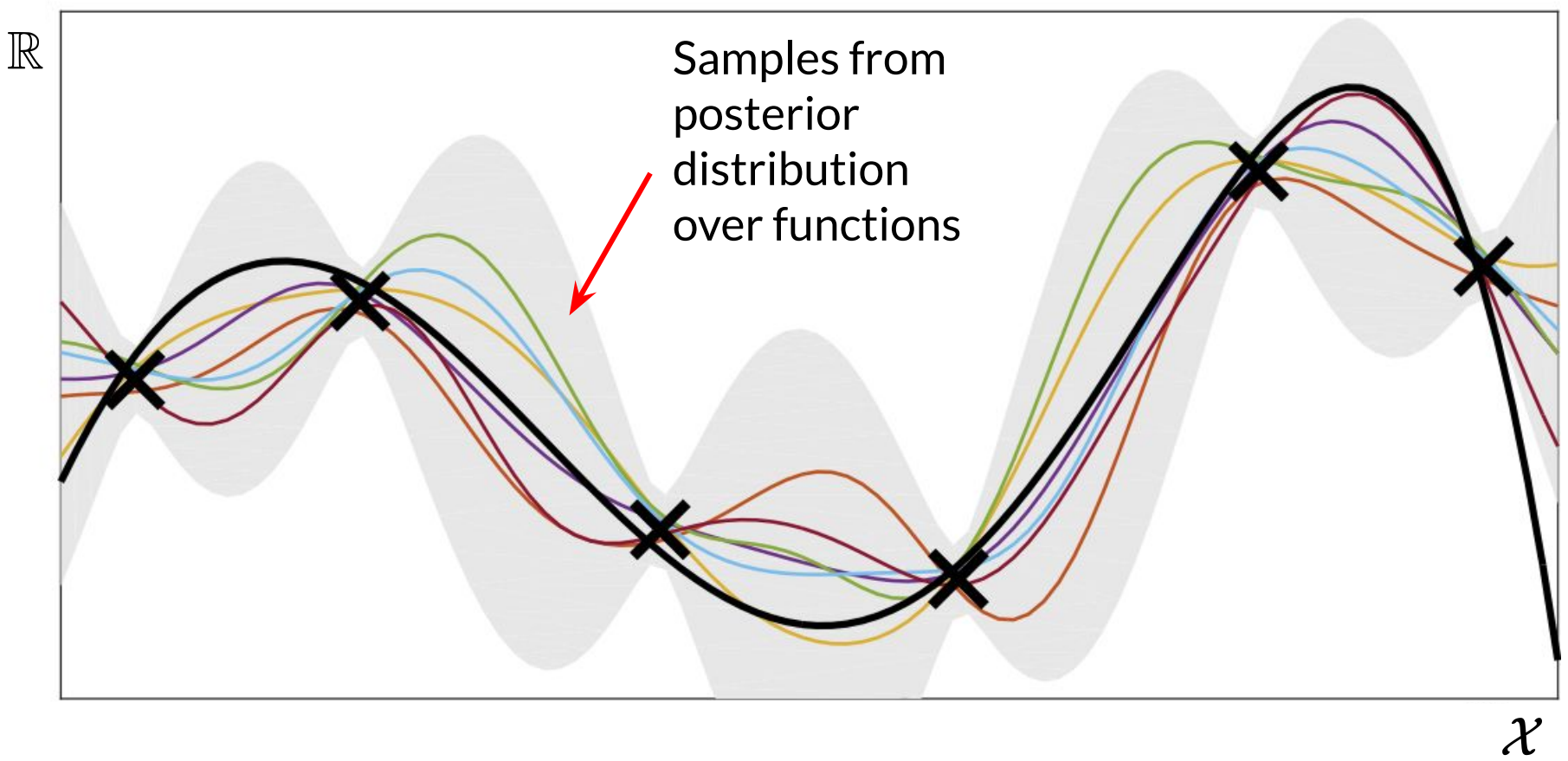
A popular choice for this model is a **Gaussian Process**.

Gaussian Processes — Intuition



A popular choice for this model is a **Gaussian Process**.

Gaussian Processes — Intuition



A popular choice for this model is a **Gaussian Process**.

Gaussian Processes — Background on Multivariate Normal

So, as mentioned a GP models the distribution over *functions* using a: **Multivariate normal (MVN) distribution**.

Most of us are familiar, but for a quick review of important details:

Gaussian Processes — Background on Multivariate Normal

Multivariate normal (MVN) distribution.

The PDF, for $x \in \mathbb{R}^d$ is:

Gaussian Processes — Background on Multivariate Normal

Multivariate normal (MVN) distribution.

The PDF, for $x \in \mathbb{R}^d$ is:

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right)$$

Gaussian Processes — Background on Multivariate Normal

Multivariate normal (MVN) distribution.

The PDF, for $x \in \mathbb{R}^d$ is:

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right)$$

With parameters:

$\mu \in \mathbb{R}^d$ — Mean vector.

$\Sigma \in \mathbb{R}^{d \times d}$ — Covariance matrix.

Gaussian Processes — Background on Multivariate Normal

Where a **covariance matrix** parameter $\sum$ is the n x n square matrix with entries $\Sigma_{ij} = \mathrm{Cov}[X_i, X_j]$, written as:

$$\Sigma = \begin{bmatrix} \mathrm{Cov}[X_1, X_1] & \cdots & \mathrm{Cov}[X_1, X_n] \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}[X_n, X_1] & \cdots & \mathrm{Cov}[X_n, X_n] \end{bmatrix}$$

Gaussian Processes — Background on Multivariate Normal

Where a **covariance matrix** parameter $\sum$ is the n x n square matrix with entries $\Sigma_{ij} = \mathrm{Cov}[X_i, X_j]$, written as:

$$\Sigma = \begin{bmatrix} \mathrm{Cov}[X_1, X_1] & \cdots & \mathrm{Cov}[X_1, X_n] \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}[X_n, X_1] & \cdots & \mathrm{Cov}[X_n, X_n] \end{bmatrix}$$

And the **covariance** of two random variables X and Y is defined to be

$$\mathrm{Cov}\left[X, Y\right] = \mathbb{E}\left[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])\right]$$

⇒ Describes whether two variables tend to move in the same direction (positive covariance) or opposite directions (negative covariance).

Gaussian Processes — Background on Multivariate Normal

Where a covariance matrix is the n x n square matrix with entries $\Sigma_{ij} = \mathrm{Cov}[X_i, X_j]$ written as

$$\Sigma = \begin{bmatrix} \mathrm{Cov}[X_1, X_1] & \cdots & \mathrm{Cov}[X_1, X_n] \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}[X_n, X_1] & \cdots & \mathrm{Cov}[X_n, X_n] \end{bmatrix}$$

Gaussian Processes — Background on Multivariate Normal

Where a covariance matrix is the n x n square matrix with entries $\Sigma_{ij} = \mathrm{Cov}[X_i, X_j]$ written as

$$\Sigma = \begin{bmatrix} \mathrm{Cov}[X_1, X_1] & \cdots & \mathrm{Cov}[X_1, X_n] \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}[X_n, X_1] & \cdots & \mathrm{Cov}[X_n, X_n] \end{bmatrix}$$

And the covariance of two random variables X and Y is defined to be

$$\mathrm{Cov}\left[X, Y\right] = \mathbb{E}\left[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])\right]$$

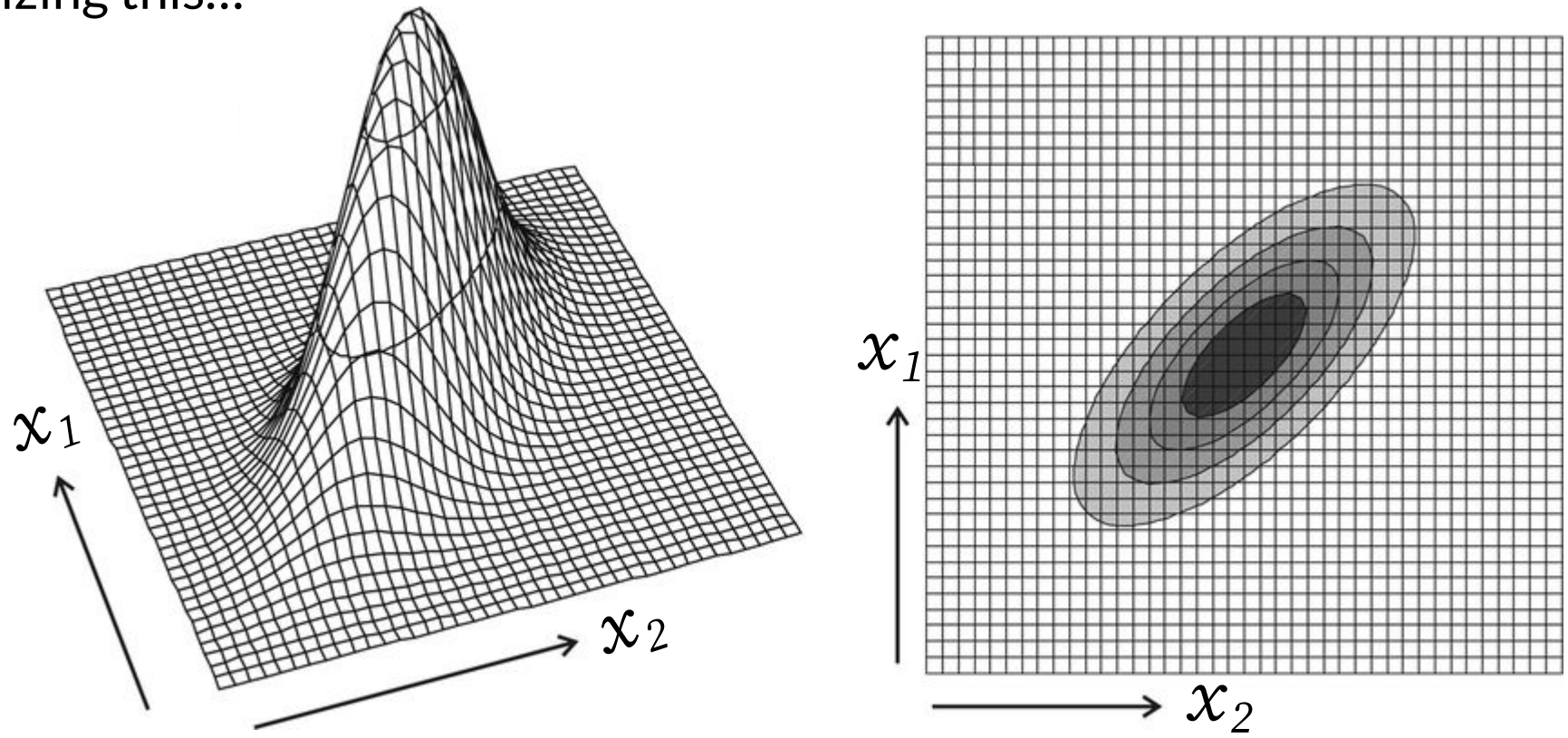
Describes whether two variables tend to move in the same direction (positive covariance) or opposite directions (negative covariance).

Gaussian Processes — Background on Multivariate Normal

Visualizing this…

Gaussian Processes — Background on Multivariate Normal

Visualizing this…

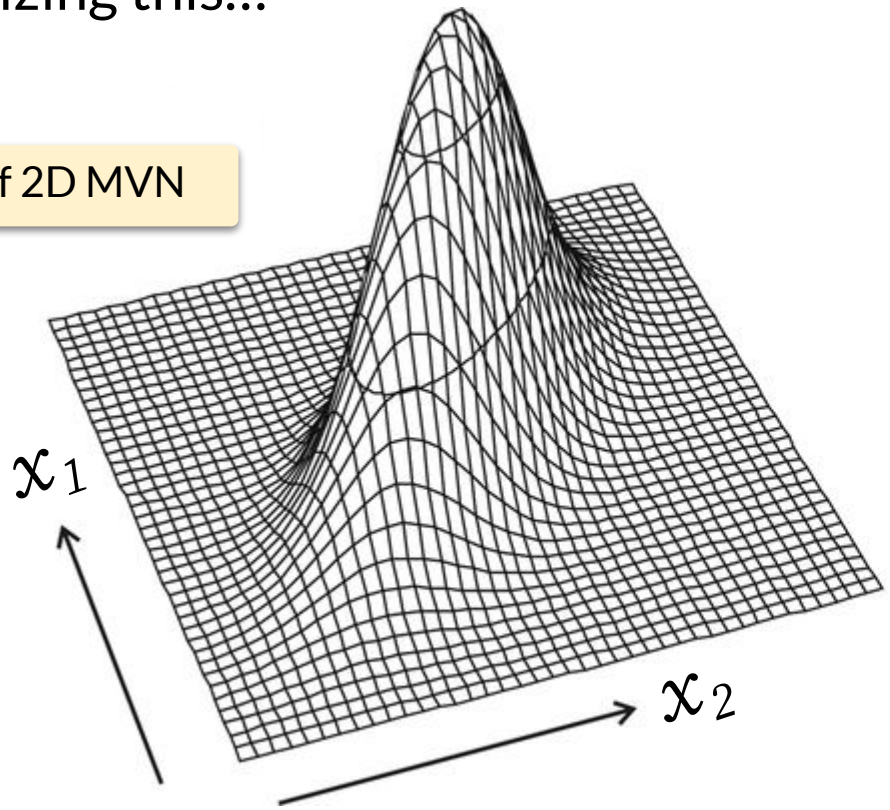


Source: Keith Sircombe, “Mountains in the shadows of time”

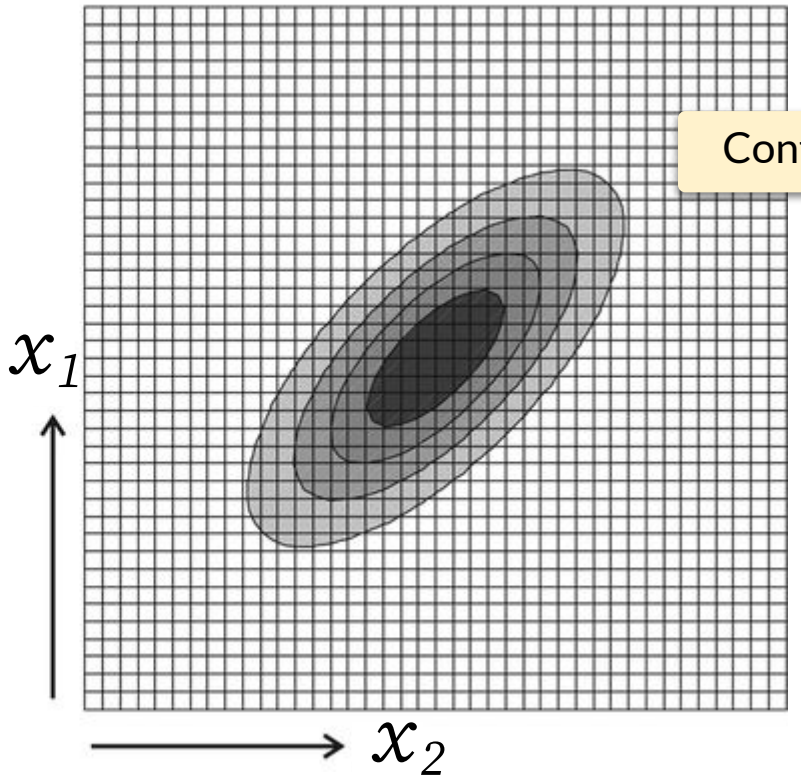
Gaussian Processes — Background on Multivariate Normal

Visualizing this…

PDF of 2D MVN



Contour Plot of PDF

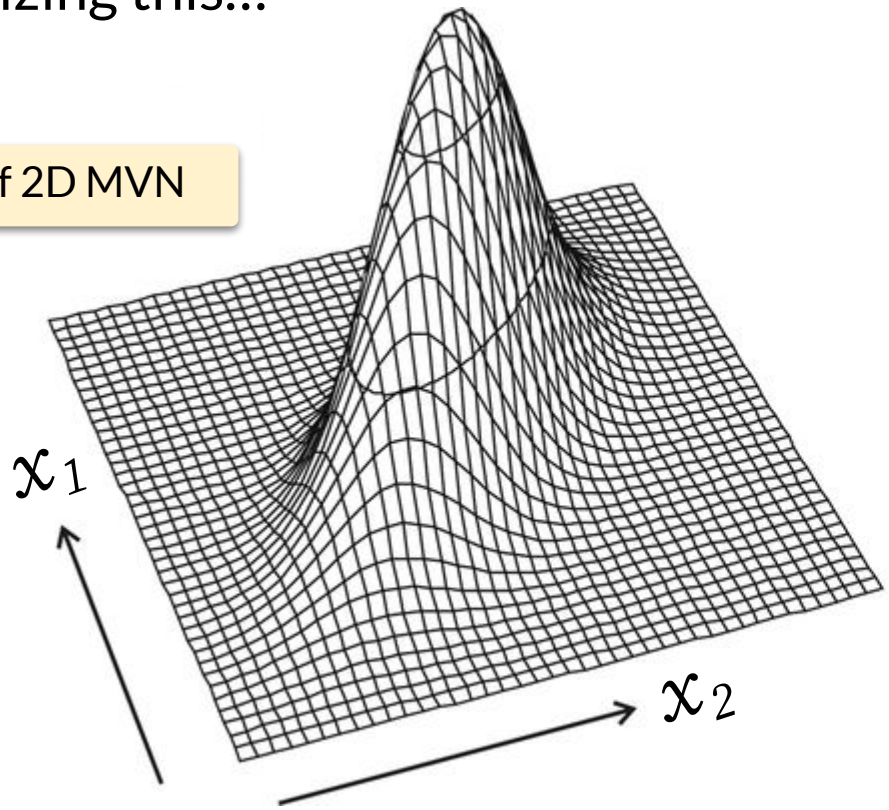


Source: Keith Sircombe, “Mountains in the shadows of time”

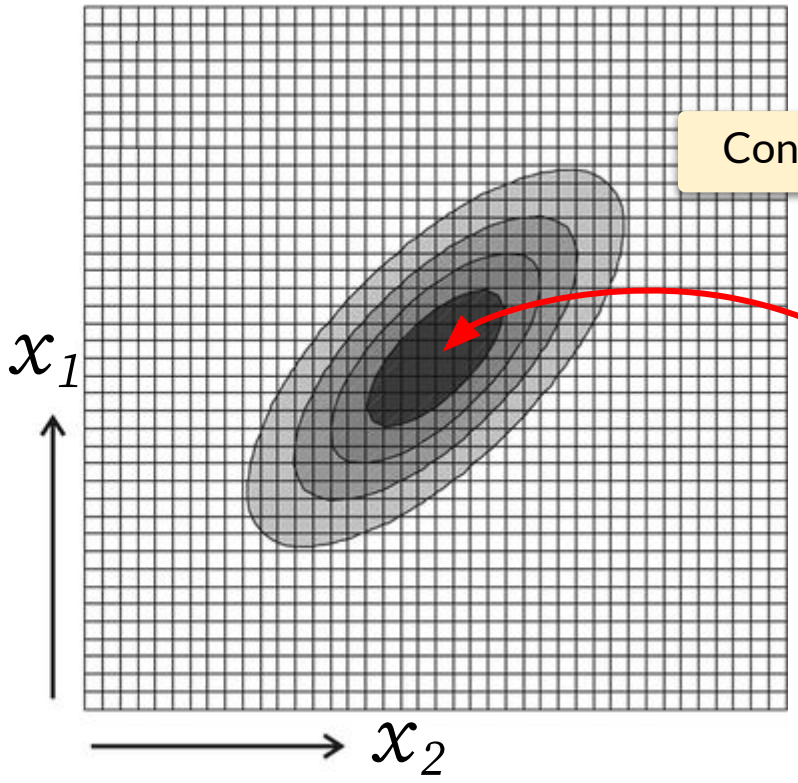
Gaussian Processes — Background on Multivariate Normal

Visualizing this…

PDF of 2D MVN



Contour Plot of PDF



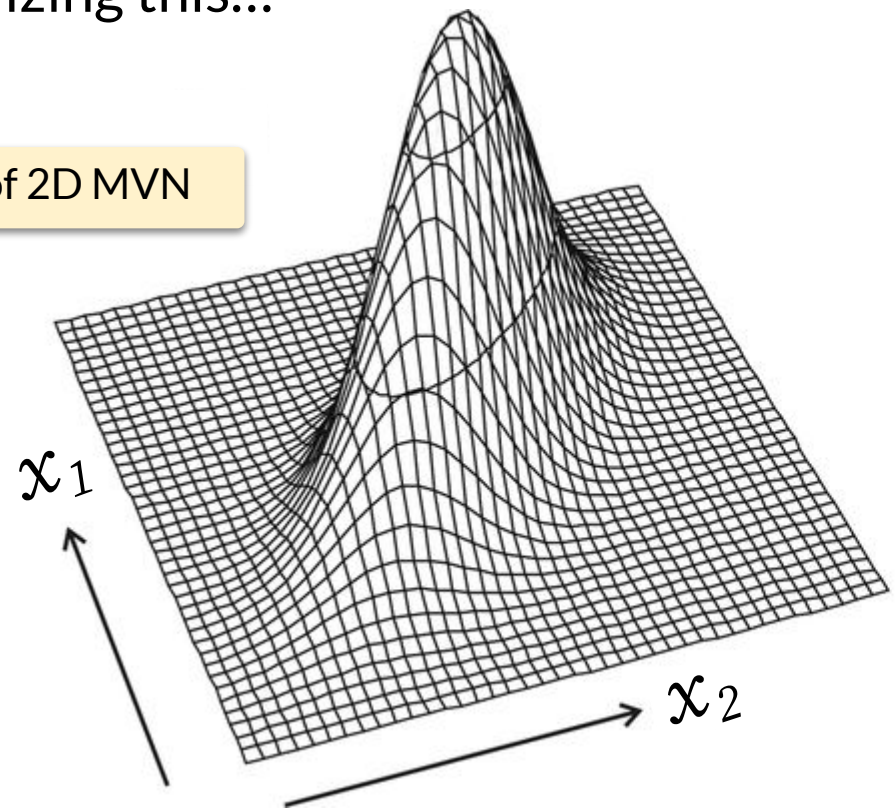
Mean

Source: Keith Sircombe, “Mountains in the shadows of time”

Gaussian Processes — Background on Multivariate Normal

Visualizing this…

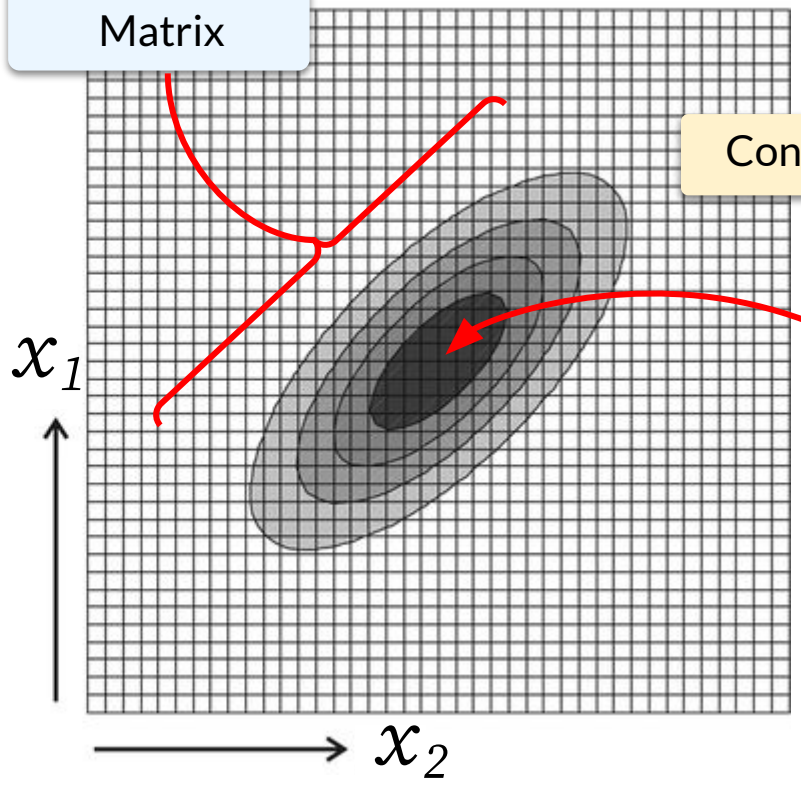
PDF of 2D MVN



Covariance Matrix

Contour Plot of PDF

Mean

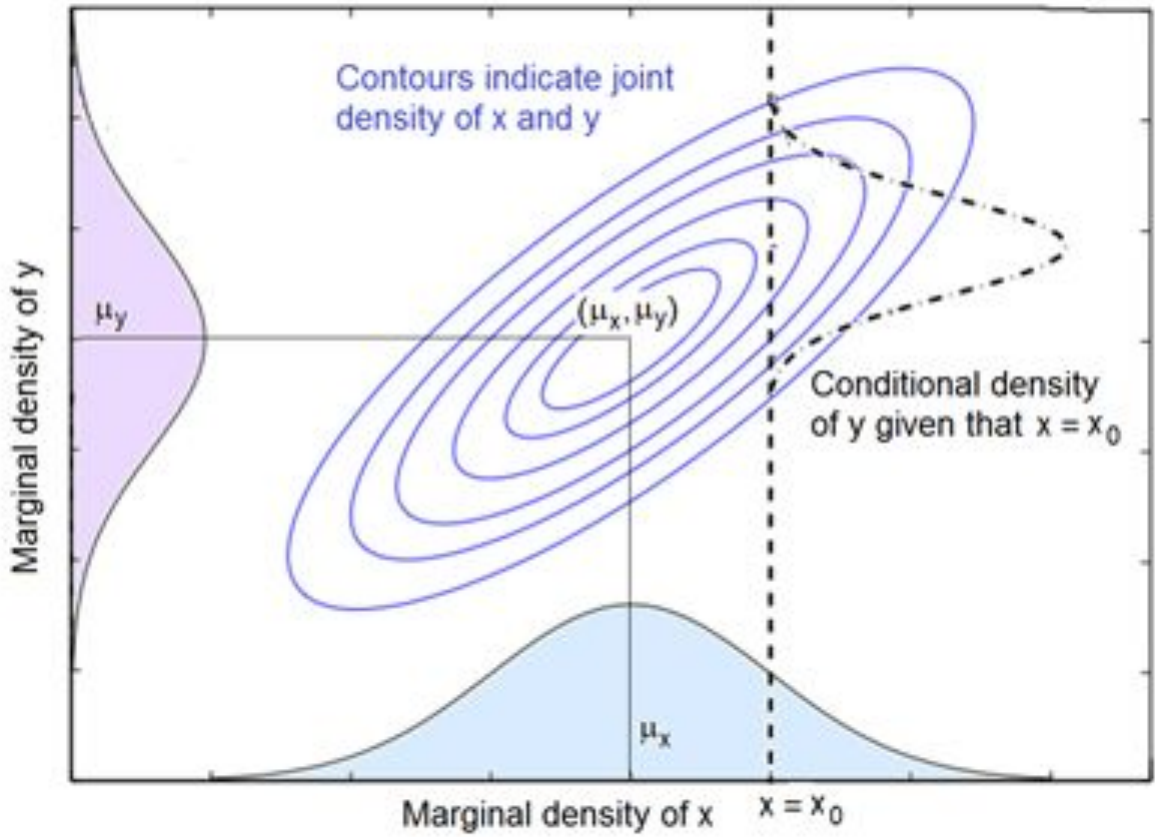


Source: Keith Sircombe, “Mountains in the shadows of time”

Gaussian Processes — Background on Multivariate Normal

Visualizing this…

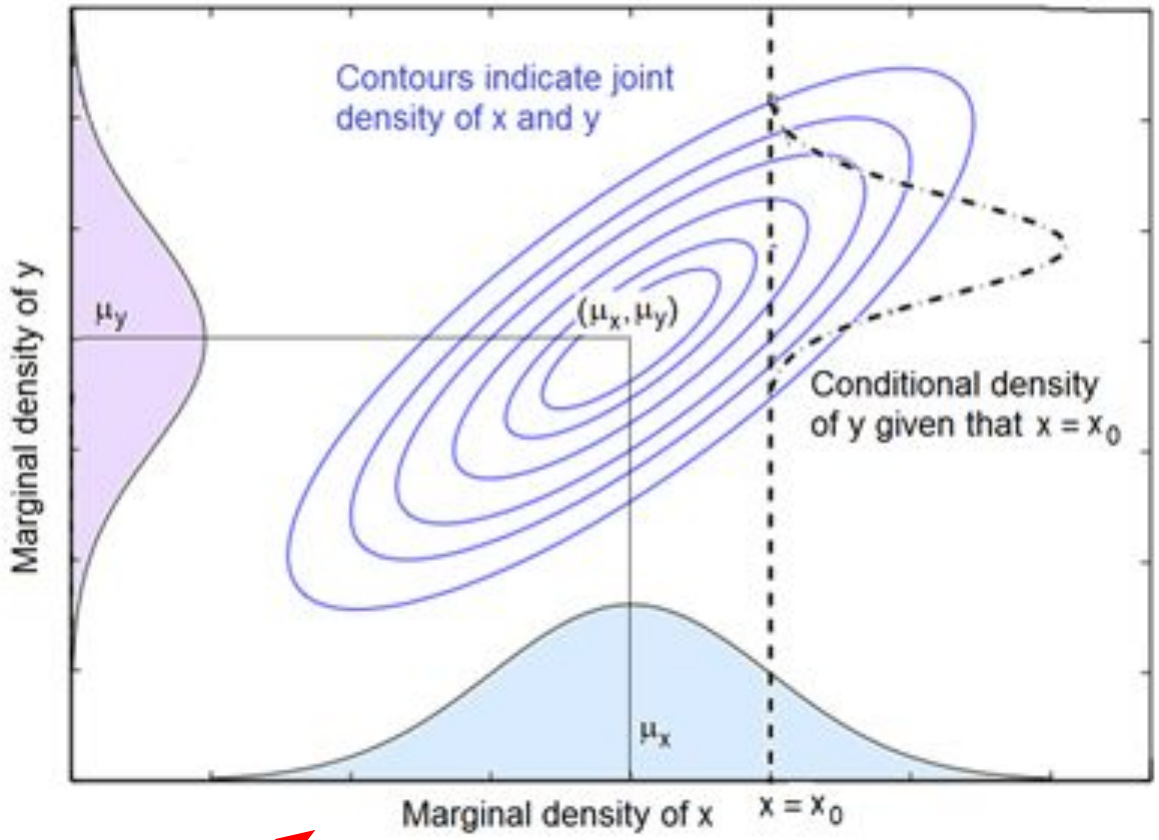
Showing marginal and conditional distributions.



Gaussian Processes — Background on Multivariate Normal

Visualizing this…

Showing marginal and conditional distributions.

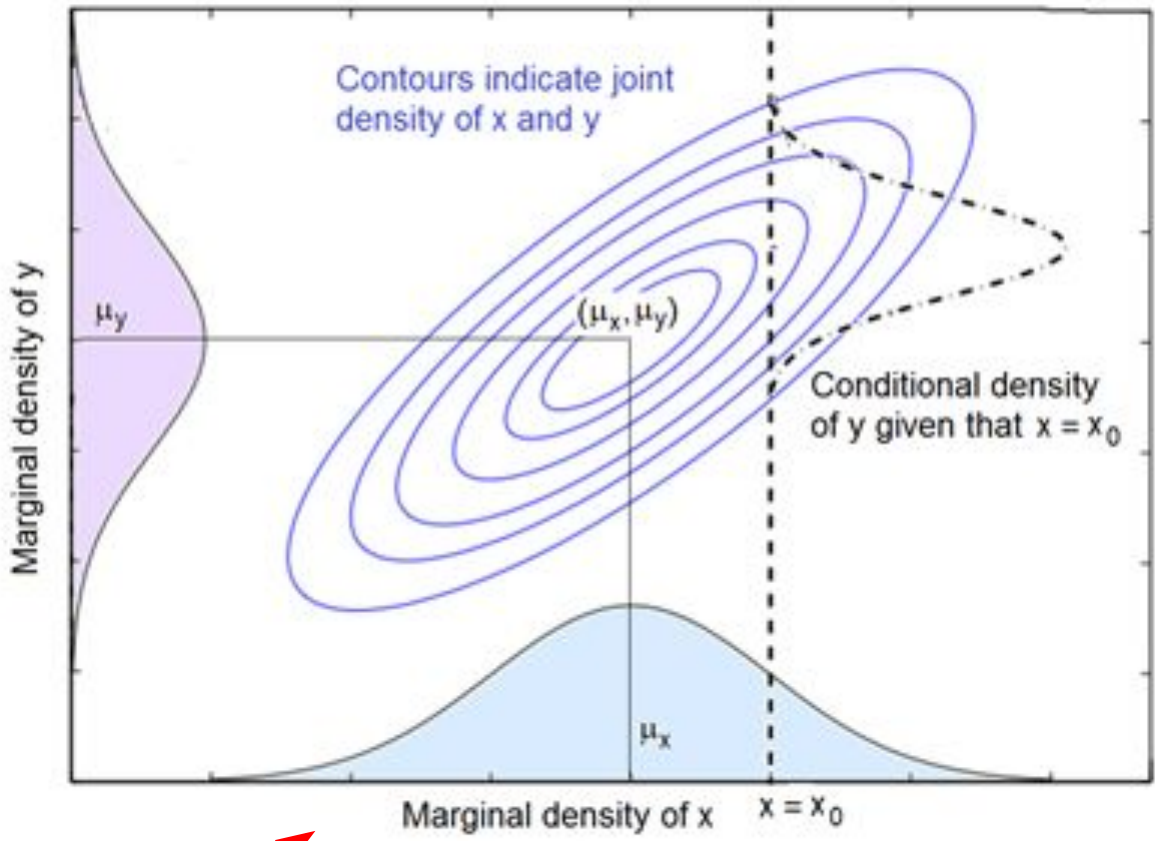


Marginal distributions are Gaussian!

Gaussian Processes — Background on Multivariate Normal

Visualizing this…

Showing marginal and conditional distributions.



Conditional distributions are also Gaussian!

Marginal distributions are Gaussian!

Gaussian Processes — More Formally

Gaussian Processes — More Formally

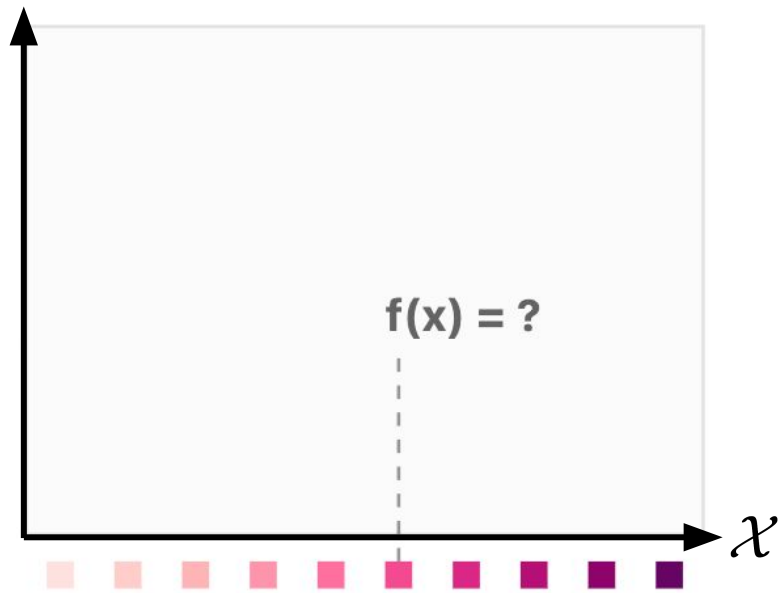
A Gaussian Process (GP) is a *stochastic process* — a collection of random variables — with the property:

> *"Every finite collection of those RVs has a multivariate normal distribution"*

What does this mean?

Gaussian Processes — More Formally

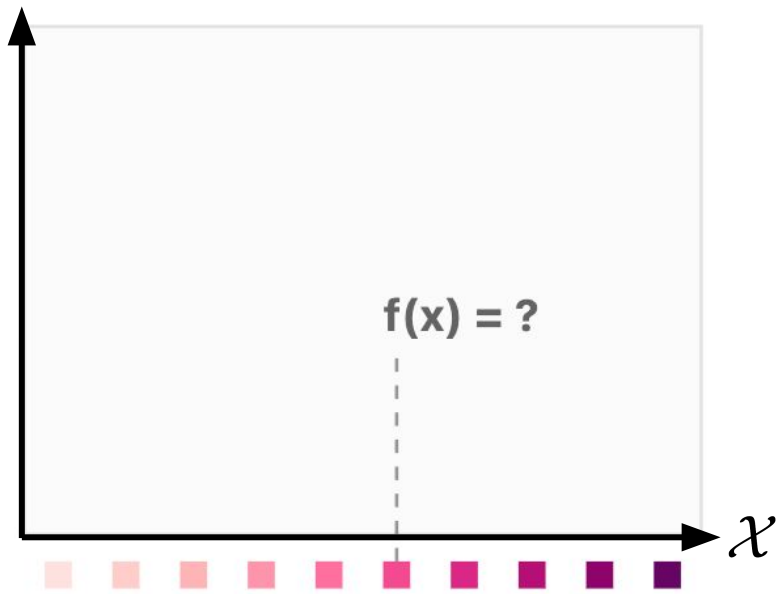
Suppose we pick a set of *d=10* points on the *x*-axis, and want to define a **function** (or set of functions) over these points:



Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — More Formally

Suppose we pick a set of *d=10* points on the *x*-axis, and want to define a **function** (or set of functions) over these points:

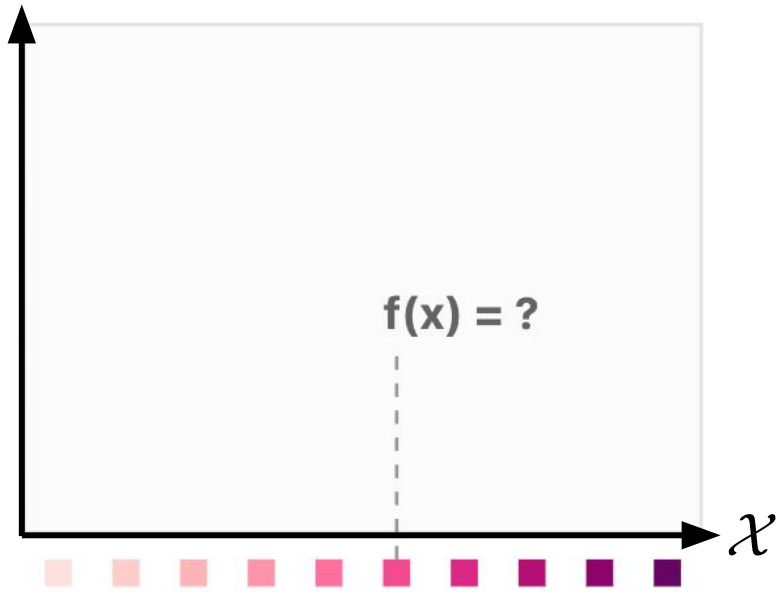


10 points are denoted by different colors

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — More Formally

Suppose we pick a set of *d=10* points on the *x*-axis, and want to define a **function** (or set of functions) over these points:



10 points are denoted by different colors

We can define a *distribution of functions*, defined on these 10 points, in the following way…

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — More Formally

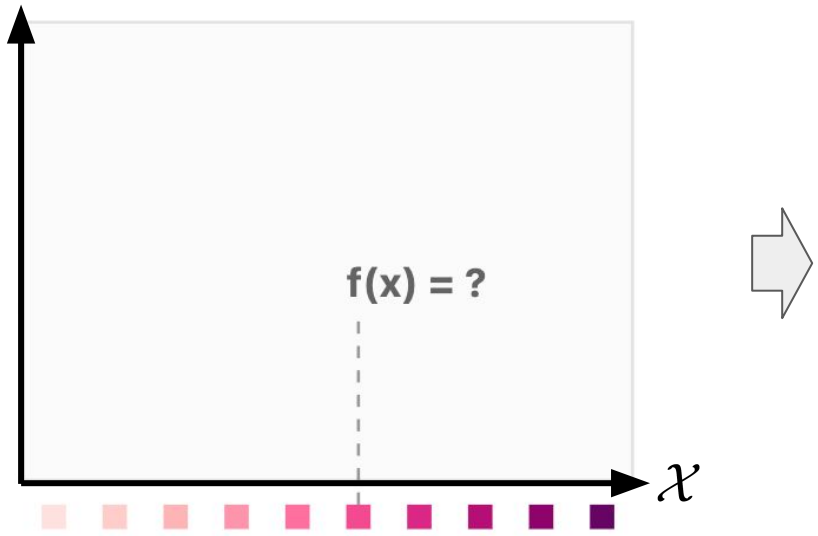
We define a MVN of ***d* dimensions**, with a mean and covariance matrix parameter.

Mean parameter $\mu \in \mathbb{R}^d$ — assume we set this to ***0*** for now.

Covariance matrix $\sum \in \mathbb{R}^{d \times d}$ — define using a *kernel function* based on distance between input points (next slide).

Gaussian Processes — More Formally

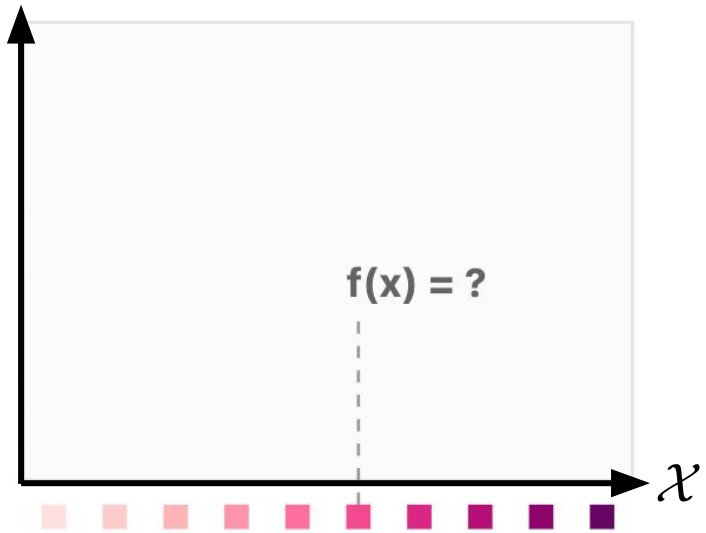
For these 10 points:



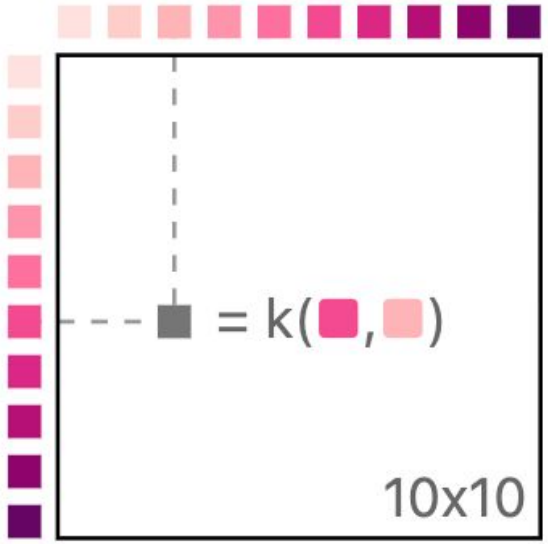
Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — More Formally

For these 10 points:



Covariance matrix:



Define covariance between any pair of inputs using kernel function $k(\cdot, \cdot)$.

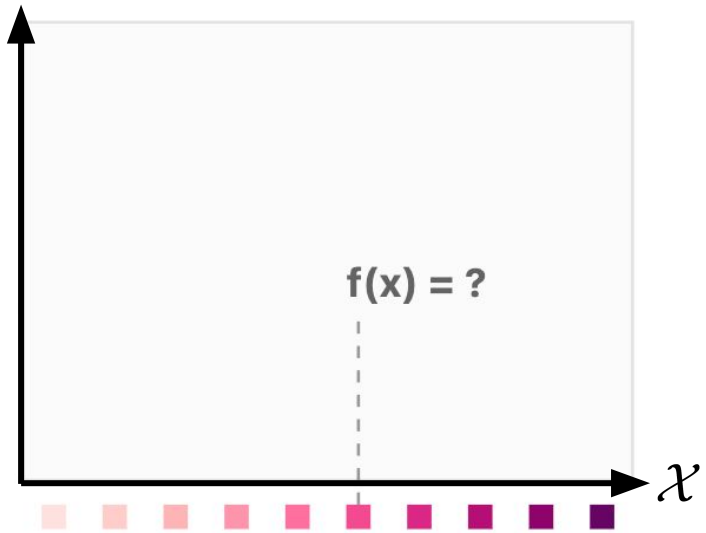
One example: *RBF kernel*,

$$k(x_1, x_2) = \sigma^2 \left(-\frac{\|x_1 - x_2\|^2}{2\ell^2} \right)$$

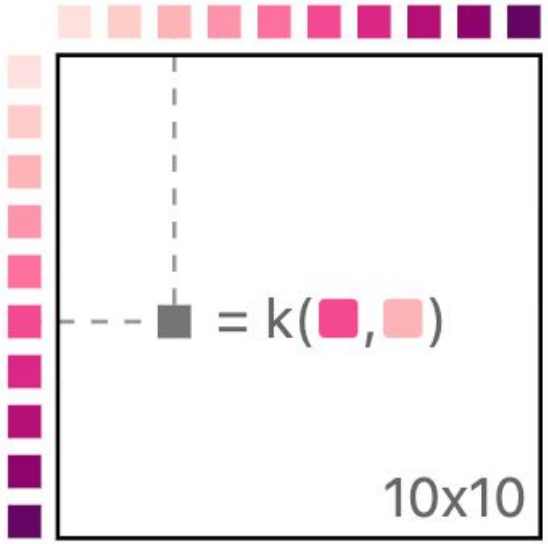
Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — More Formally

For these 10 points:



Covariance matrix:



Define covariance between any pair of inputs using kernel function $k(\cdot, \cdot)$.

One example: *RBF kernel*,

$$k(x_1, x_2) = \sigma^2 \left(-\frac{\|x_1 - x_2\|^2}{2\ell^2} \right)$$

Highest when $x_1 = x_2$, lowest when x_2 very different from x_2.

Source: A Visual Exploration of Gaussian Processes

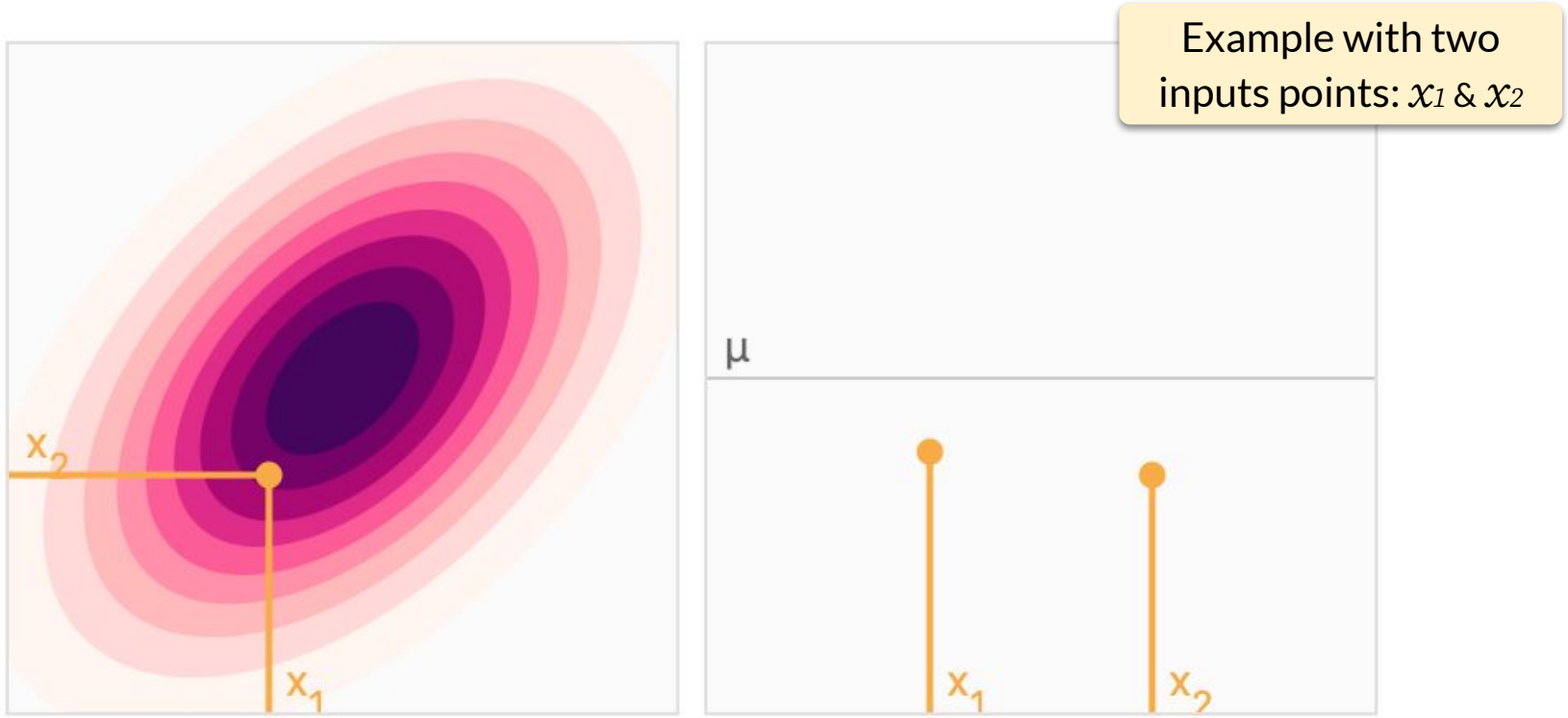
Gaussian Processes — More Formally

This yields a MVN, where samples give function values for the *d* input points:

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — More Formally

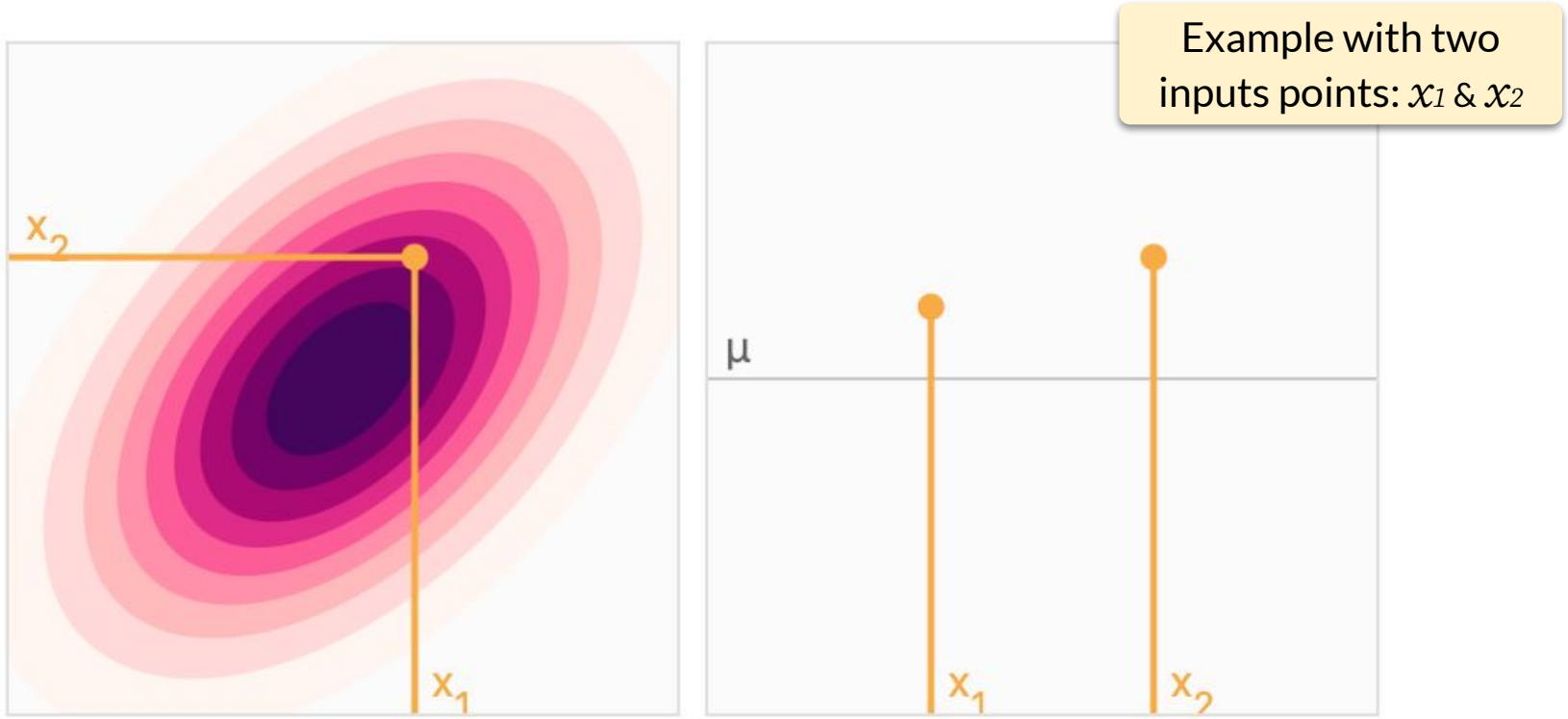
This yields a MVN, where samples give function values for the *d* input points:



Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — More Formally

This yields a MVN, where samples give function values for the *d* input points:

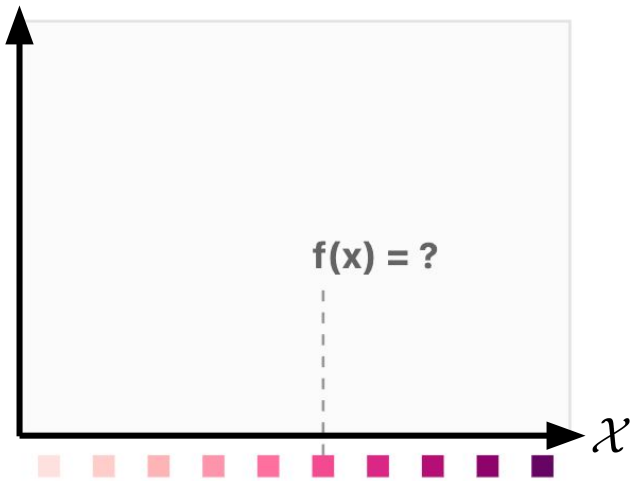


Source: A Visual Exploration of Gaussian Processes

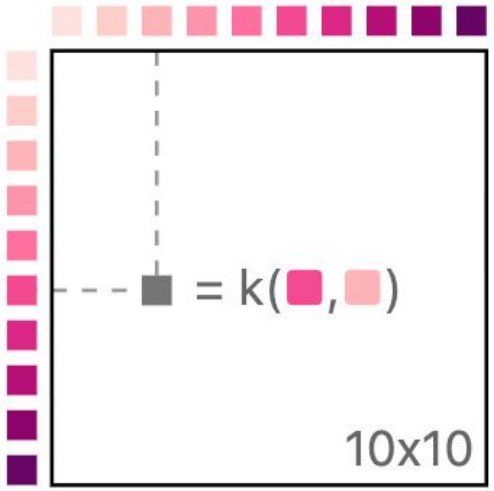
Gaussian Processes — More Formally

This yields a MVN, where samples give function values for the *d* input points:

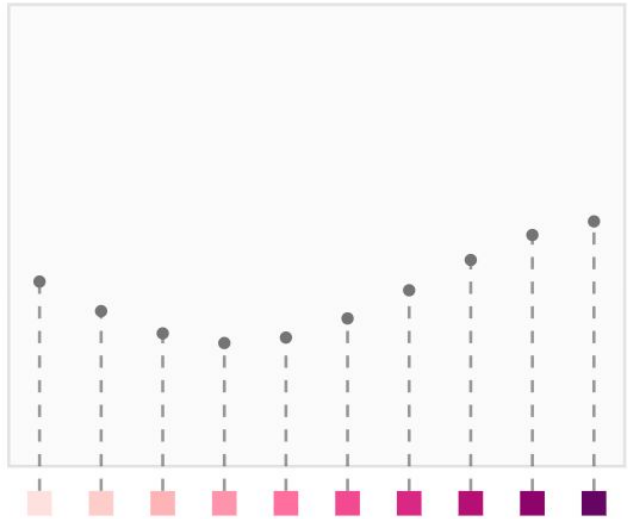
For these 10 points:



Covariance matrix:



Sampled function values



Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — More Formally

All together:

Gaussian Processes — More Formally

All together:

We define a **Gaussian Process (GP)** as a distribution over functions, parameterized by a *mean function* $m(\cdot)$ and *covariance kernel function* $k(\cdot, \cdot)$, written

$$f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$$

Gaussian Processes — More Formally

All together:

We define a **Gaussian Process (GP)** as a distribution over functions, parameterized by a *mean function* $m(\cdot)$ and *covariance kernel function* $k(\cdot, \cdot)$, written

$$f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$$

Where, for any finite collection of *d* input points, we can:

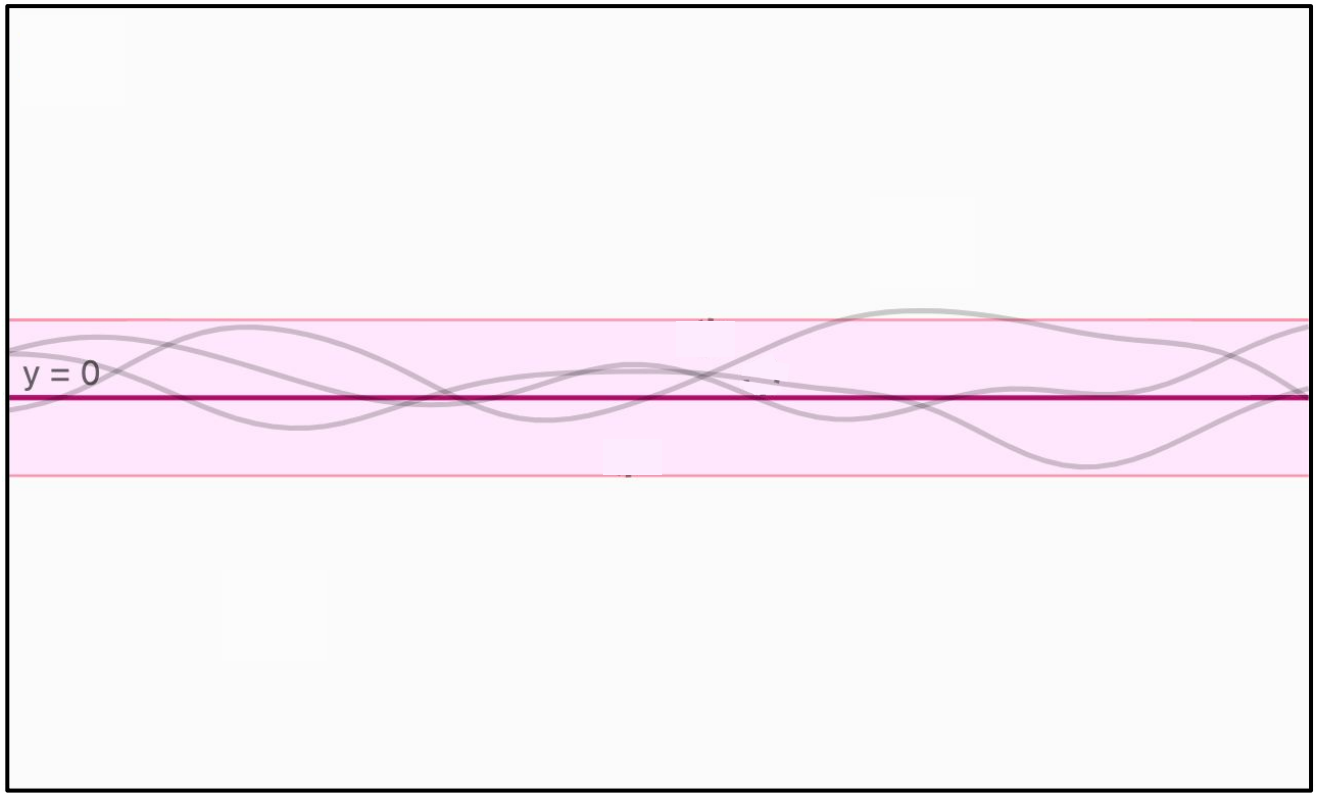
- Define a MVN *on these d points*, using the mean/kernel functions to define a mean parameter and covariance matrix.
- And this MVN yields a distribution over function values.

Gaussian Processes — Sausage Plot

What does this distribution look like?

Gaussian Processes — Sausage Plot

What does this distribution look like? → **often drawn like this:**

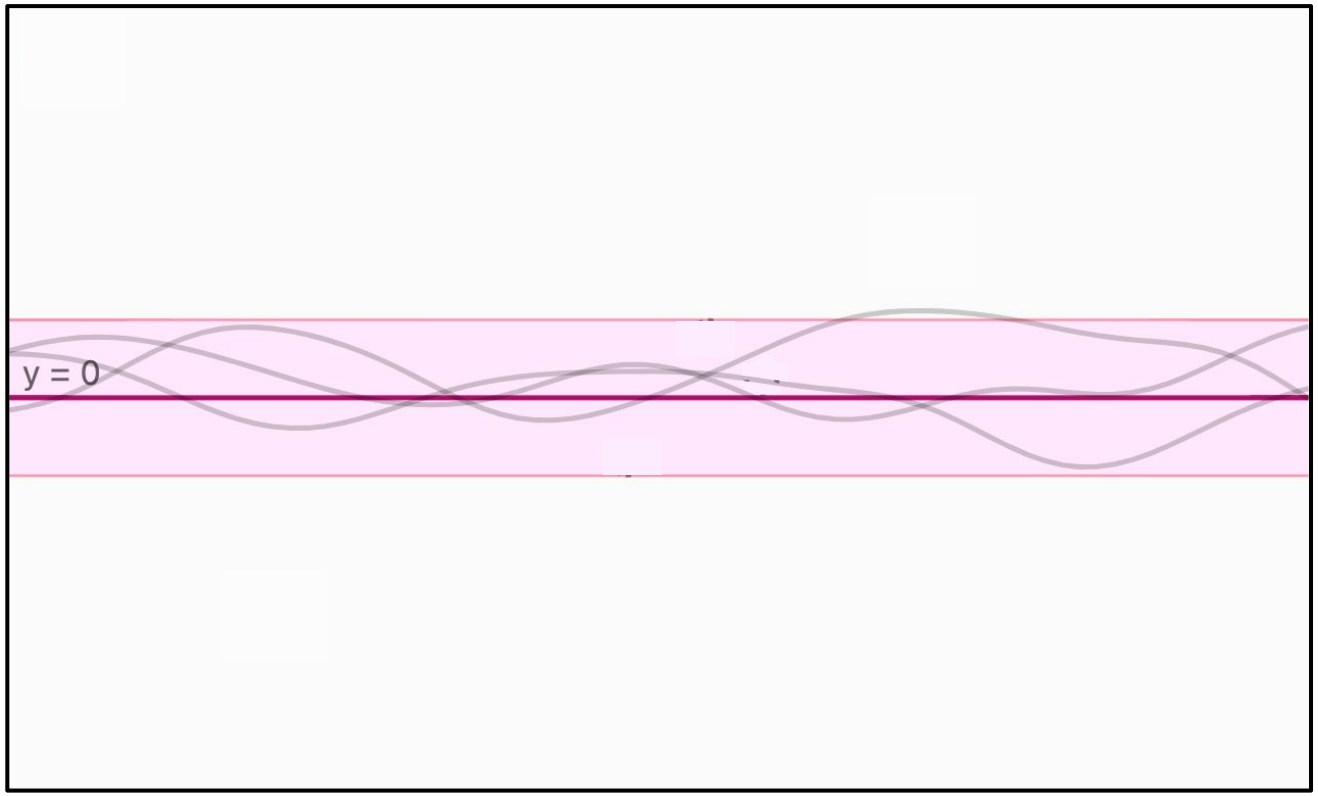


Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Sausage Plot

Called a *Sausage Plot* because it looks like a sausage

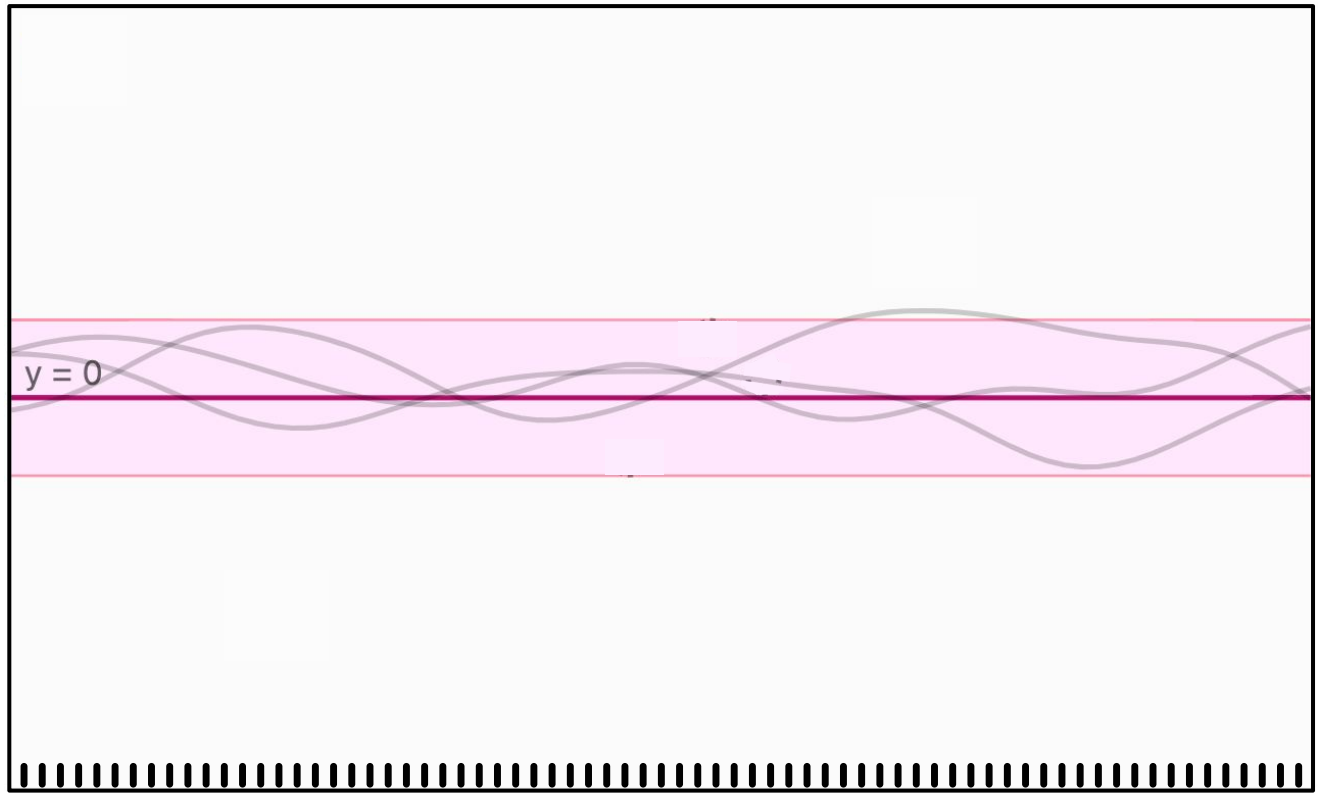
What does this distribution look like? → **often drawn like this:**



Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Sausage Plot

What does this distribution look like? → **often drawn like this:**

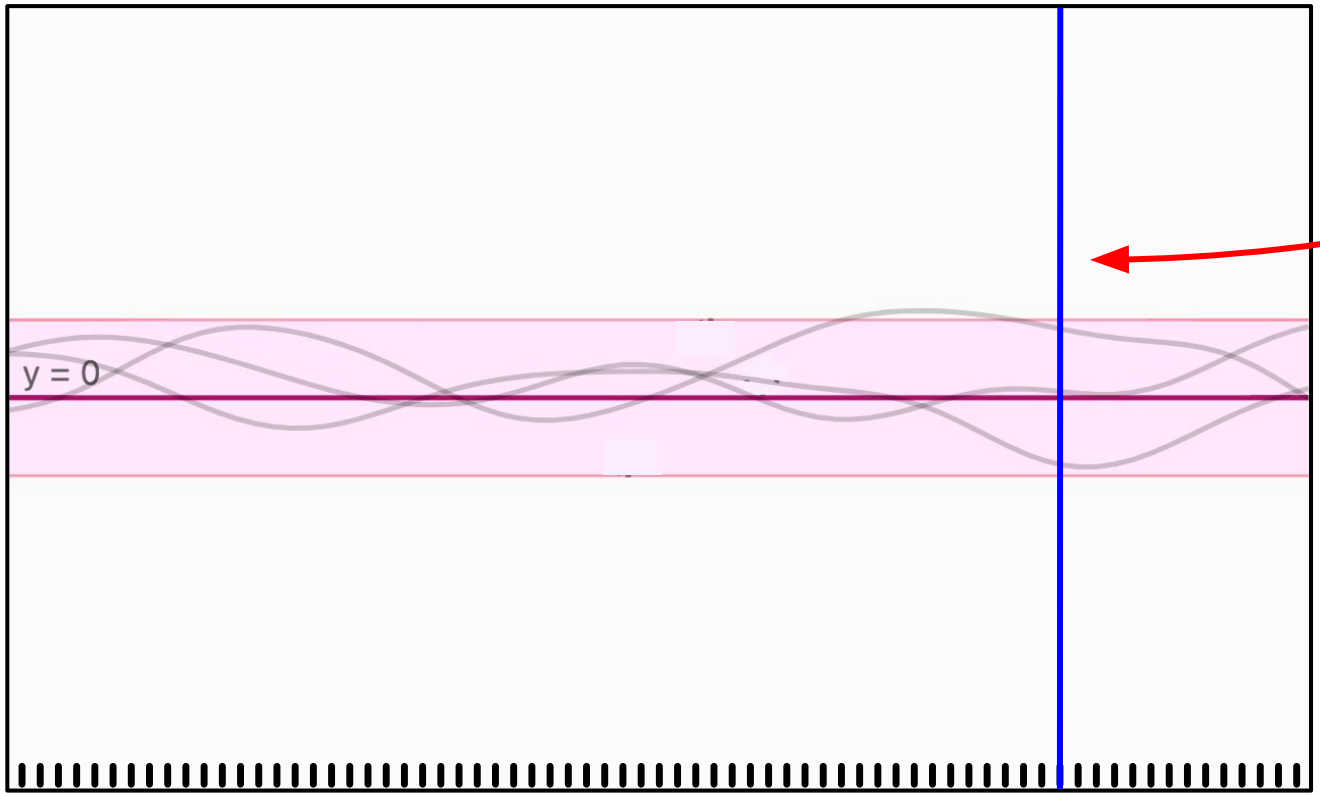


This plot depicts a MVN defined on a large set of grid points

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Sausage Plot

What does this distribution look like? → **often drawn like this:**



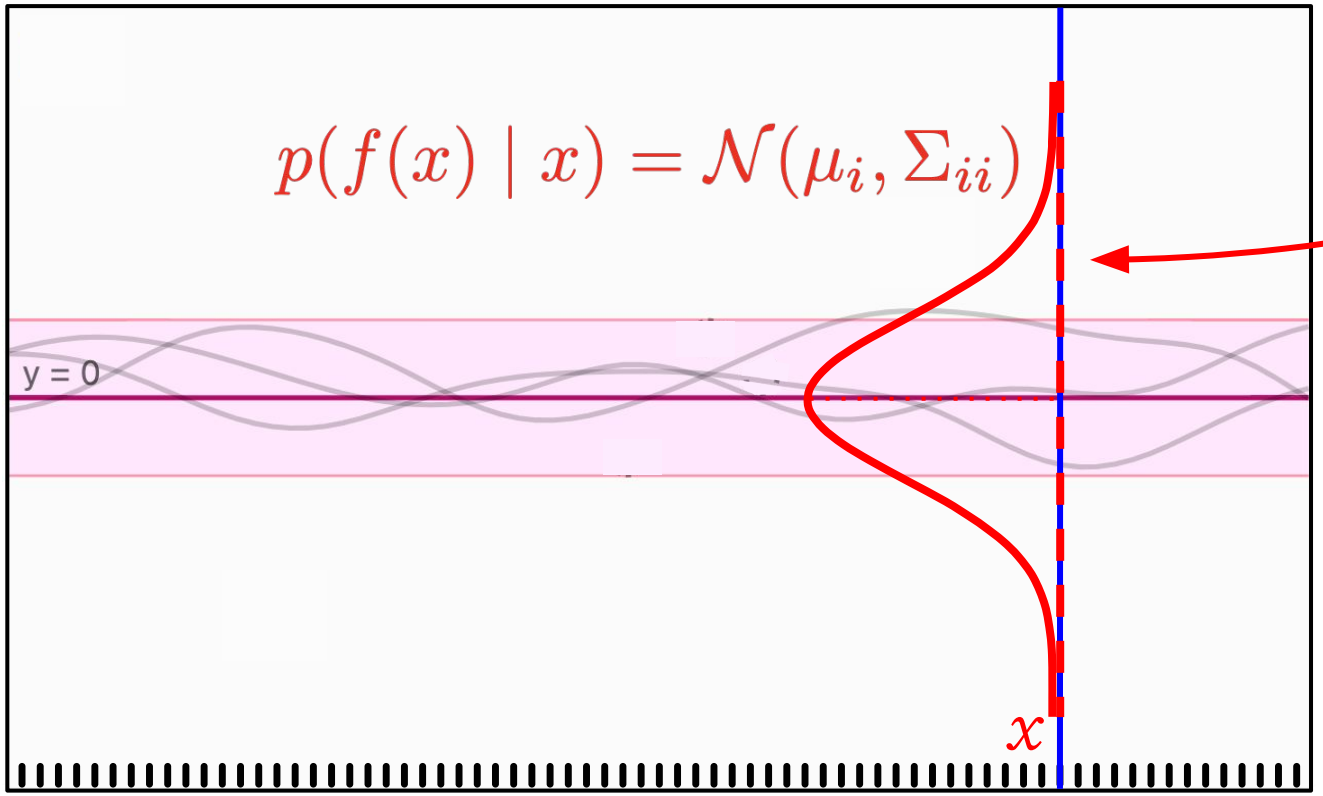
For any grid point, can compute a marginal distribution of MVN

This plot depicts a MVN defined on a large set of grid points

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Sausage Plot

What does this distribution look like? → **often drawn like this:**



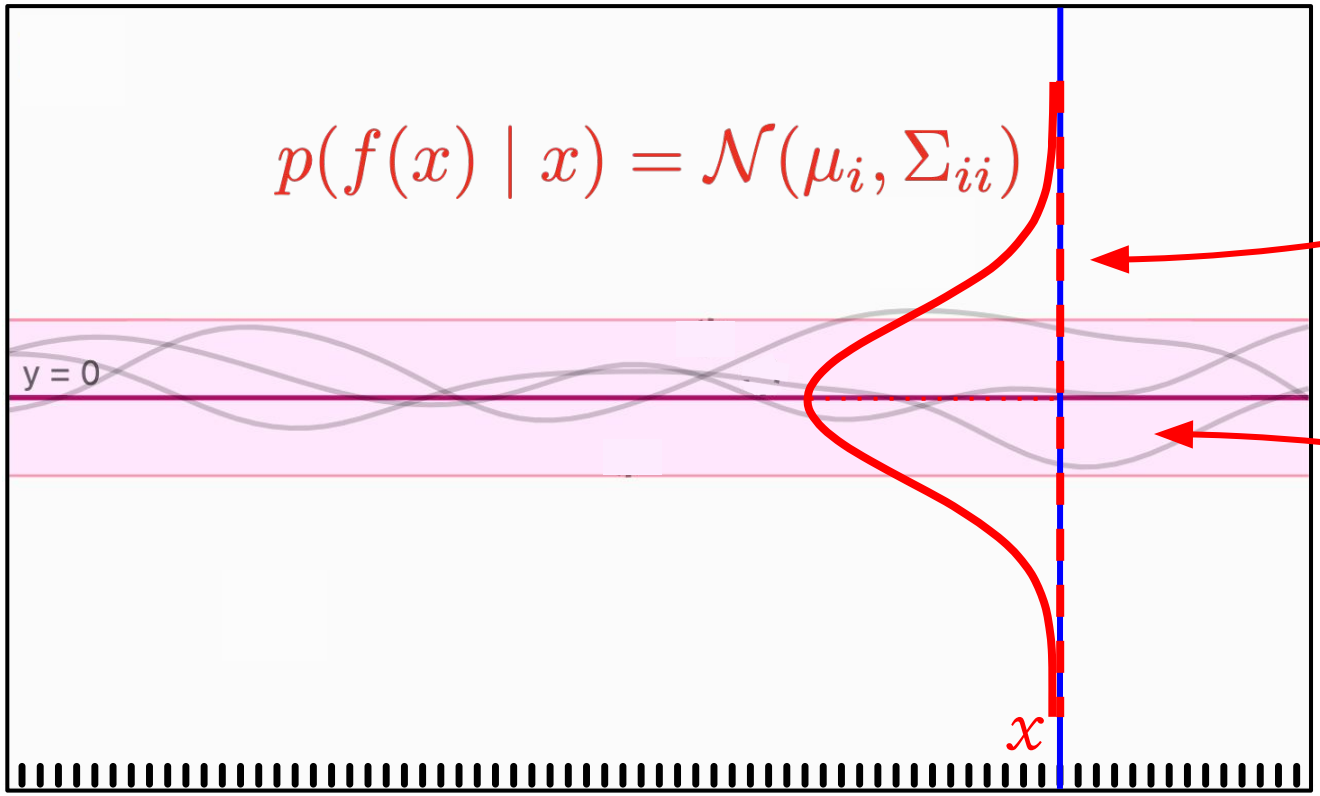
For any grid point, can compute a marginal distribution of MVN

This plot depicts a MVN defined on a large set of grid points

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Sausage Plot

What does this distribution look like? → **often drawn like this:**



For any grid point, can compute a marginal distribution of MVN

This plot depicts a MVN defined on a large set of grid points

Samples from the MVN are shown in grey (appear as functions!)

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Posterior

So this lets us define a distribution over functions on any collection of inputs points.

Next, how do we use GPs for predictive UQ models?

Gaussian Processes — Posterior

So this lets us define a distribution over functions on any collection of inputs points.

Next, how do we use GPs for predictive UQ models?

- *I.e.*, we have a dataset of (x, y) pairs.
- And we want to learn a predictive model, and then make predictions (along with uncertainty) on new test inputs.

Gaussian Processes — Posterior

So this lets us define a distribution over functions on any collection of inputs points.

Next, how do we use GPs for predictive UQ models?

- *I.e.*, we have a dataset of (x, y) pairs.
- And we want to learn a predictive model, and then make predictions (along with uncertainty) on new test inputs.

⇒ We will do Bayesian inference using this model!

Gaussian Processes — Posterior

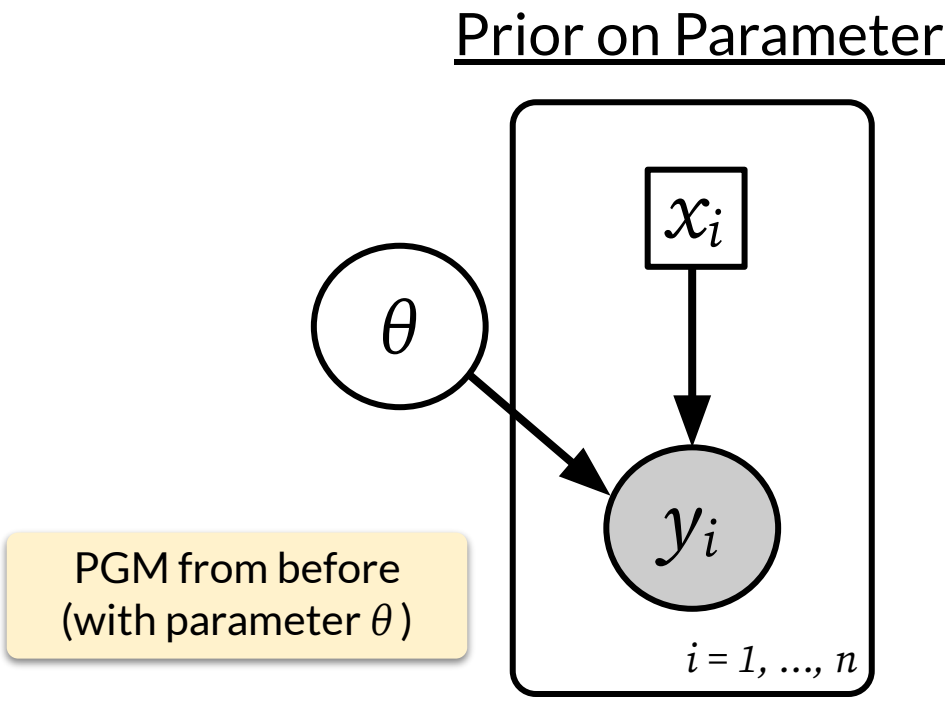
In particular, we will:

- Use a GP to define a prior over functions.
- Observe a set of (x, y) pairs.
- Compute a posterior GP over functions.

Gaussian Processes — Posterior

In particular, we will:

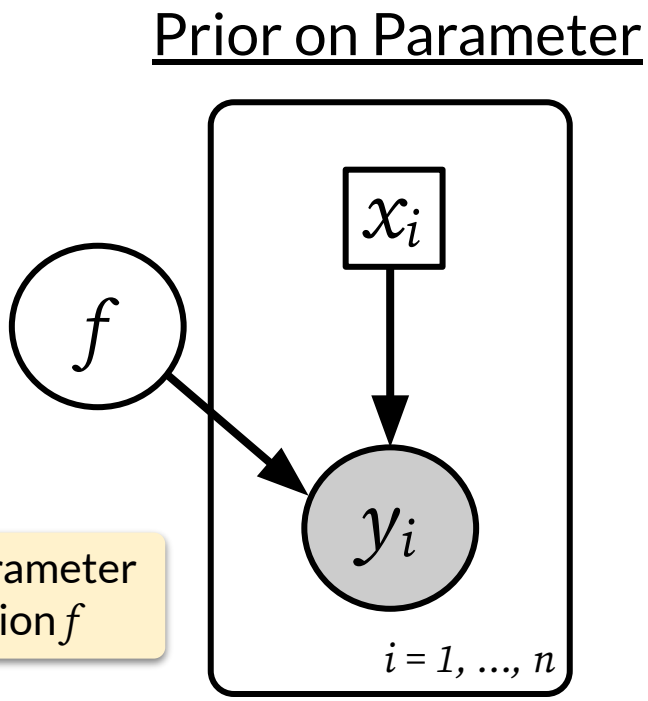
- Use a GP to define a prior over functions.
- Observe a set of (x, y) pairs.
- Compute a posterior GP over functions.



Gaussian Processes — Posterior

In particular, we will:

- Use a GP to define a prior over functions.
- Observe a set of (x, y) pairs.
- Compute a posterior GP over functions.



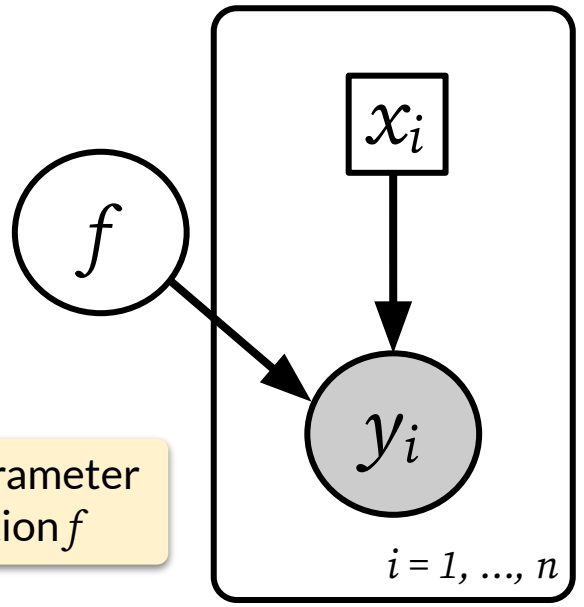
In GPs, the parameter is the function f

Gaussian Processes — Posterior

In particular, we will:

- Use a GP to define a prior over functions.
- Observe a set of (x, y) pairs.
- Compute a posterior GP over functions.

Prior on Parameter



In GPs, the parameter is the function f

We want GP posterior:
$p(f \mid \{y_i\})$

Gaussian Processes — Posterior

Computing the posterior:

Gaussian Processes — Posterior

Computing the posterior:

- We have a GP prior *(essentially a multivariate normal)* : $f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$

Gaussian Processes — Posterior

Computing the posterior:

- We have a GP prior *(essentially a multivariate normal)* : $f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$
- We have a Gaussian likelihood for observations: $y \sim p(y \mid f\ ;\ x) = \mathcal{N}(f, \sigma^2)$

Gaussian Processes — Posterior

Computing the posterior:

- We have a GP prior *(essentially a multivariate normal)* : $f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$
- We have a Gaussian likelihood for observations: $y \sim p(y \mid f\ ;\ x) = \mathcal{N}(f, \sigma^2)$
- ⇒ Posterior is *(essentially)* also a multivariate normal distribution.

Gaussian Processes — Posterior

Computing the posterior:

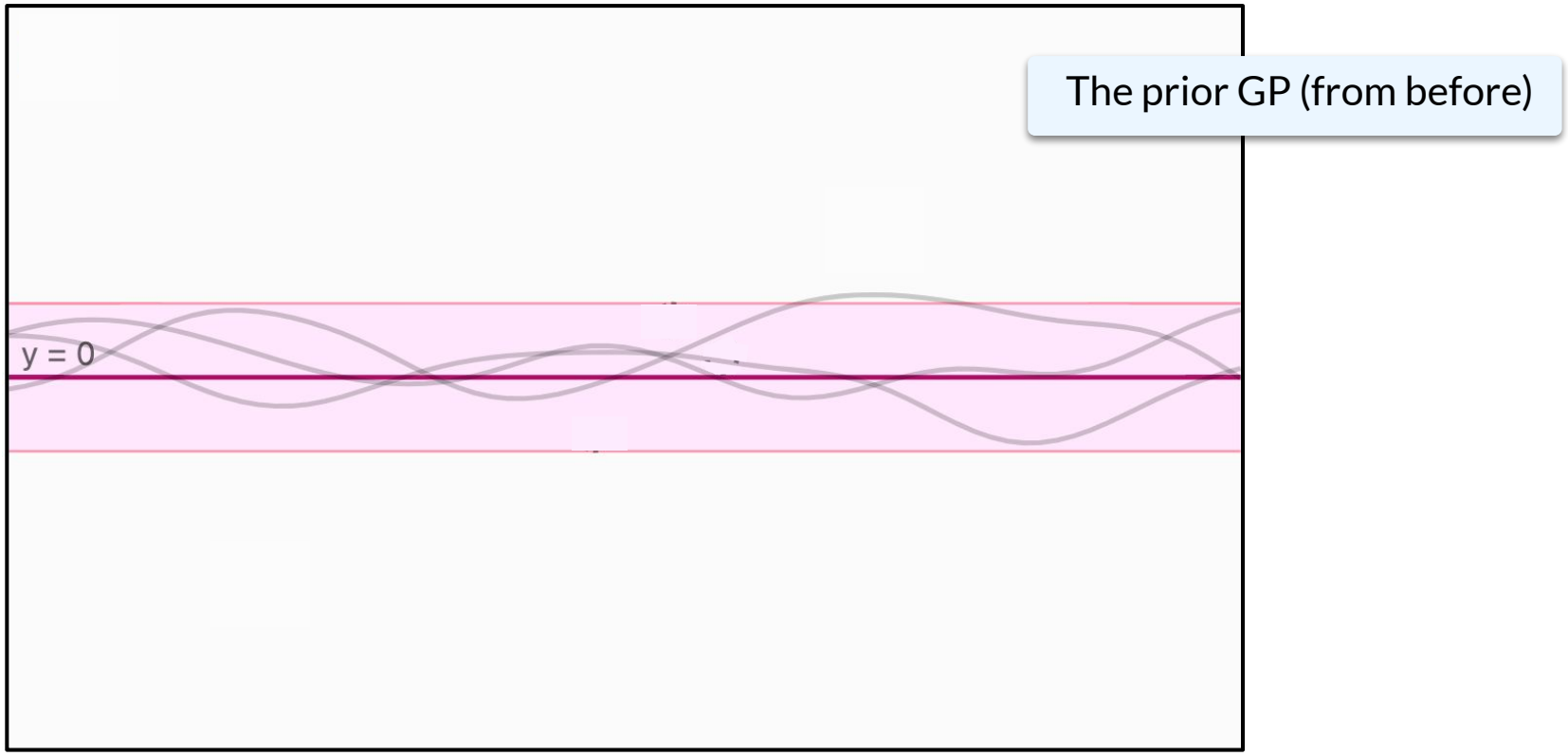
- We have a GP prior *(essentially a multivariate normal)* : $f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$
- We have a Gaussian likelihood for observations: $y \sim p(y \mid f\ ;\ x) = \mathcal{N}(f, \sigma^2)$
- ⇒ Posterior is *(essentially)* also a multivariate normal distribution.

Posterior is technically a **posterior Gaussian process**, so it's another GP with a mean function and covariance kernel parameters

⇒ which can be computed in closed form.

Gaussian Processes — Posterior

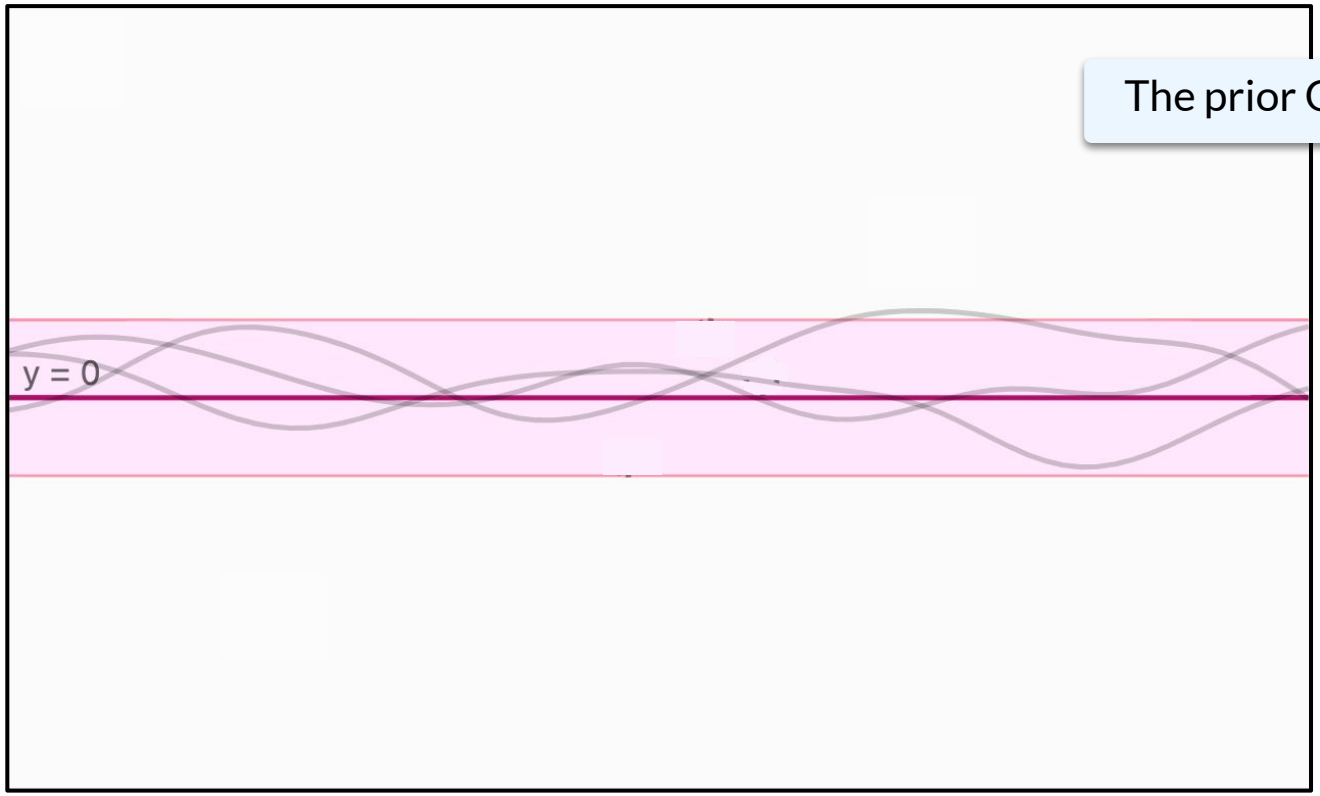
Visualizing this:



Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Posterior

Visualizing this:



The prior GP (from before)

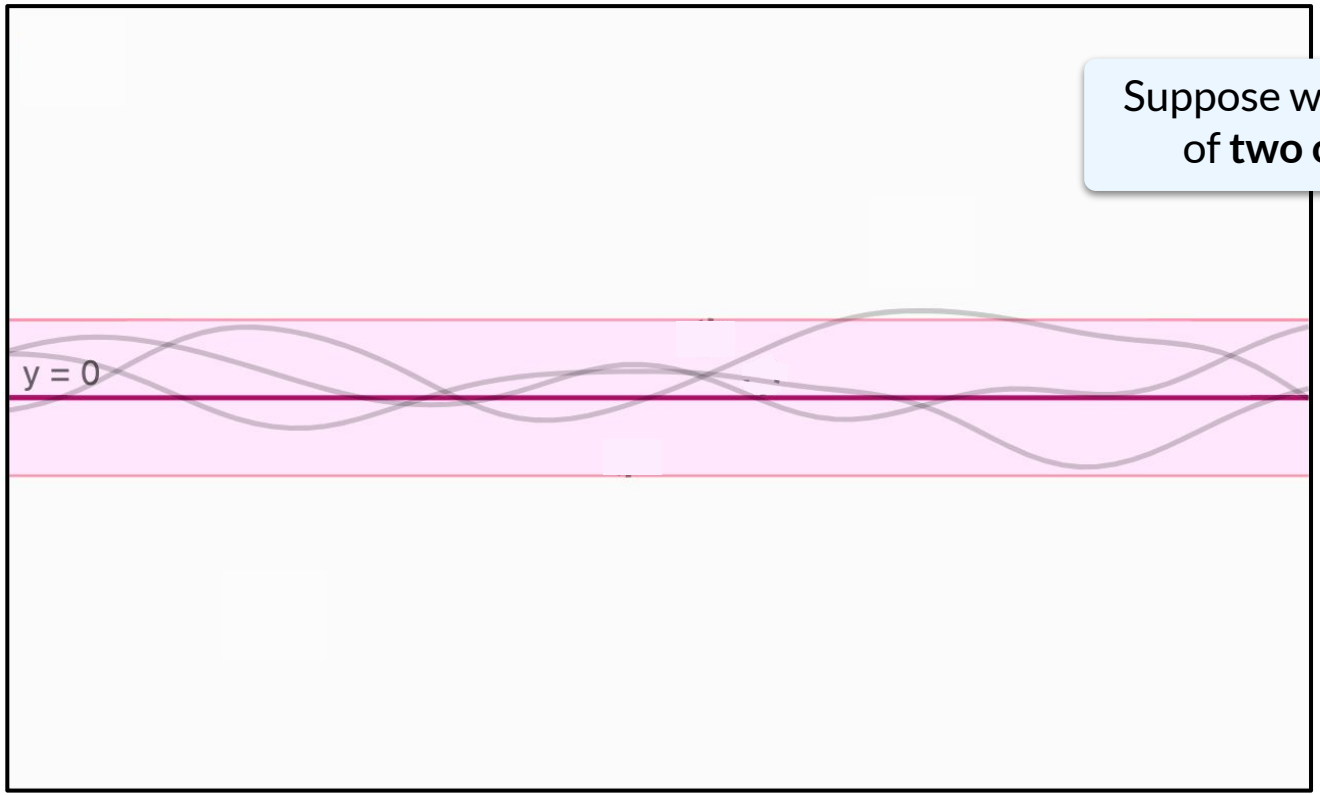
Mean function is constant **0** function

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Posterior

Visualizing this:

Mean function is constant **0** function

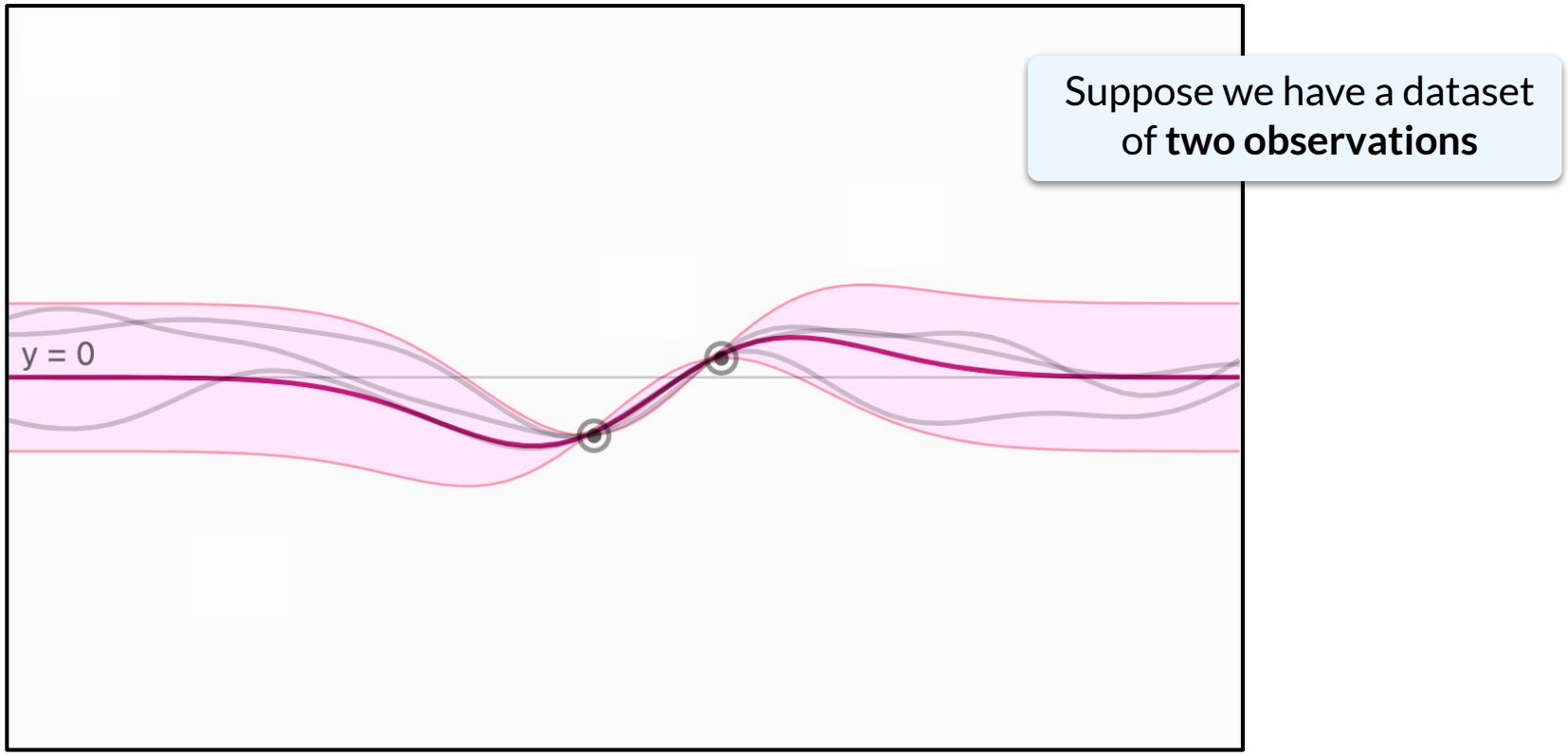


Suppose we have a dataset of **two observations**

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Posterior

Visualizing this:

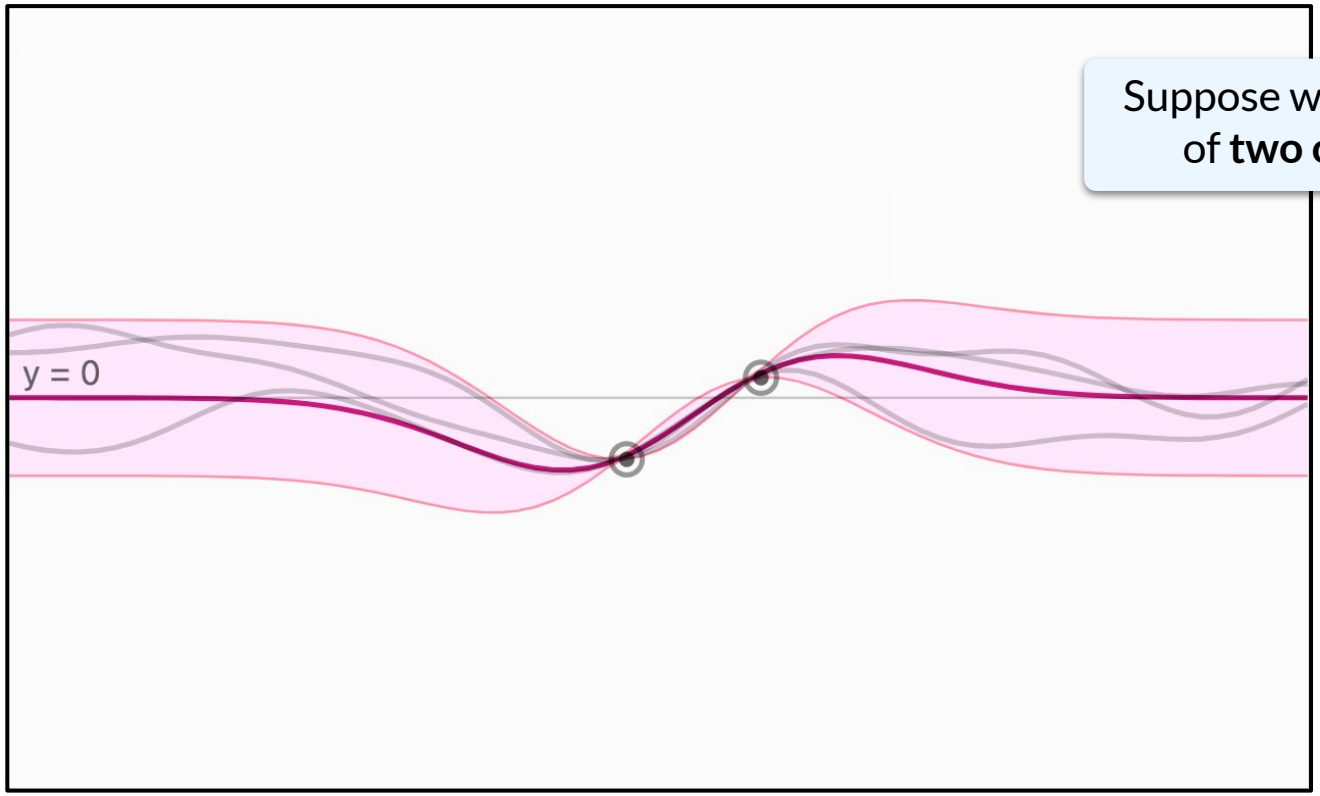


Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Posterior

Visualizing this:

Mean function is now **not constant**

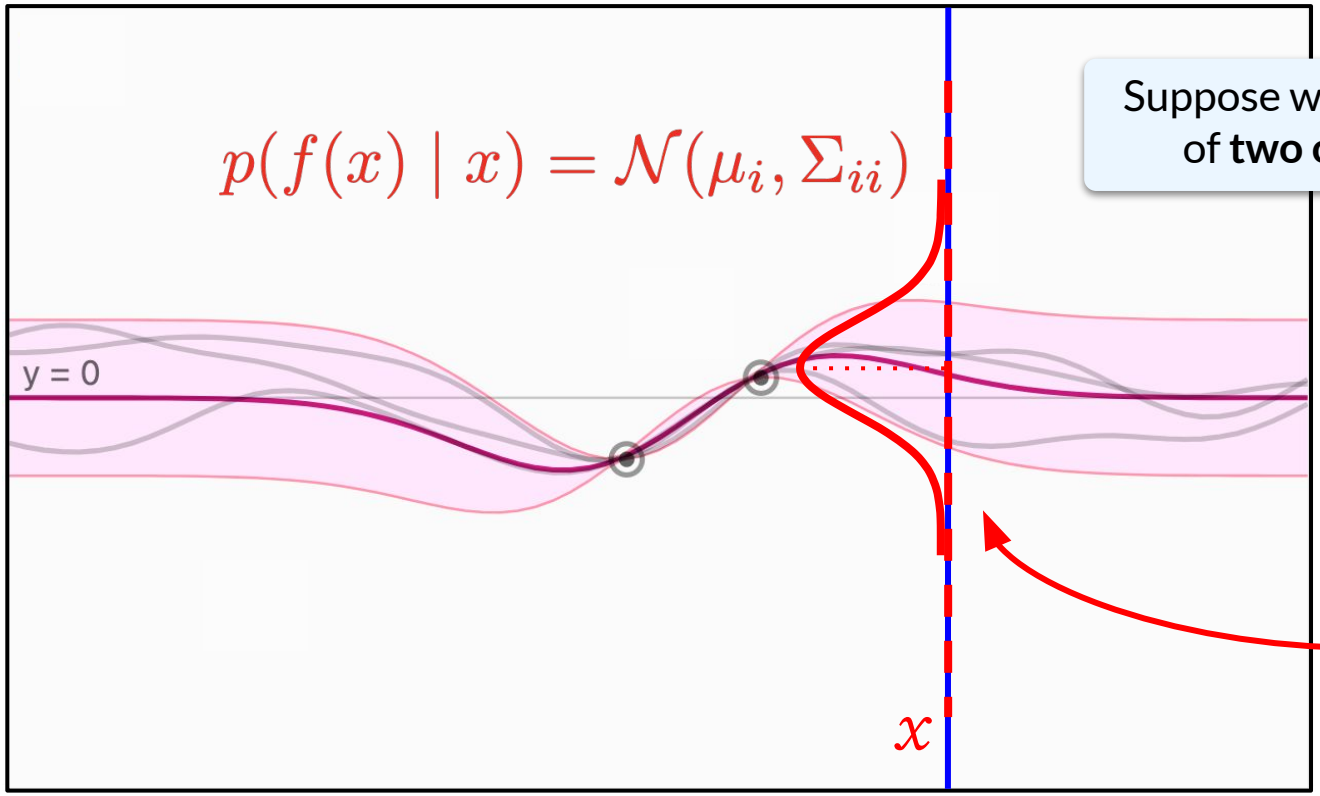


Suppose we have a dataset of **two observations**

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Posterior

Visualizing this:



Suppose we have a dataset of **two observations**

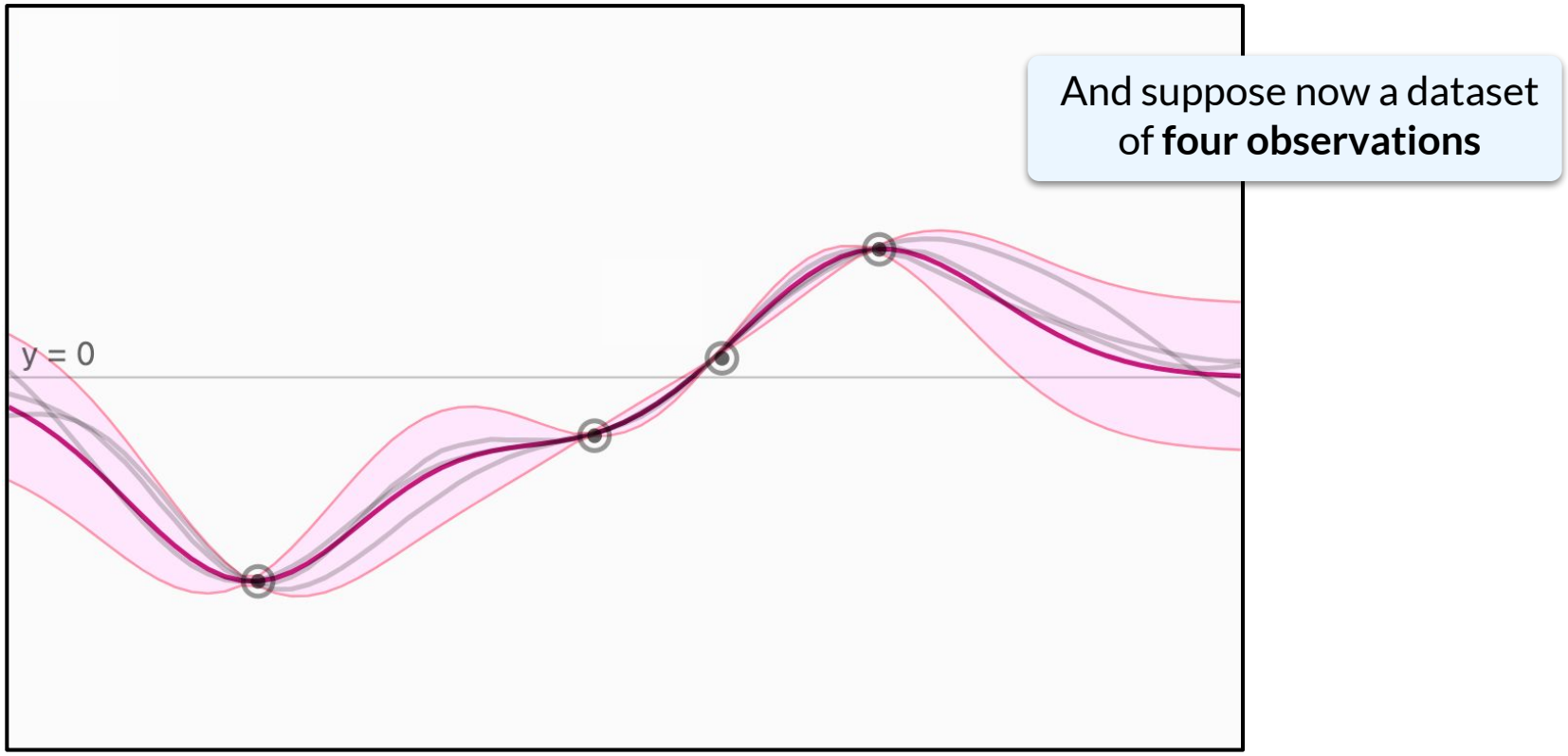
Mean function is now **not constant**

To make predictions for a given x, just compute marginal of posterior GP

Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Posterior

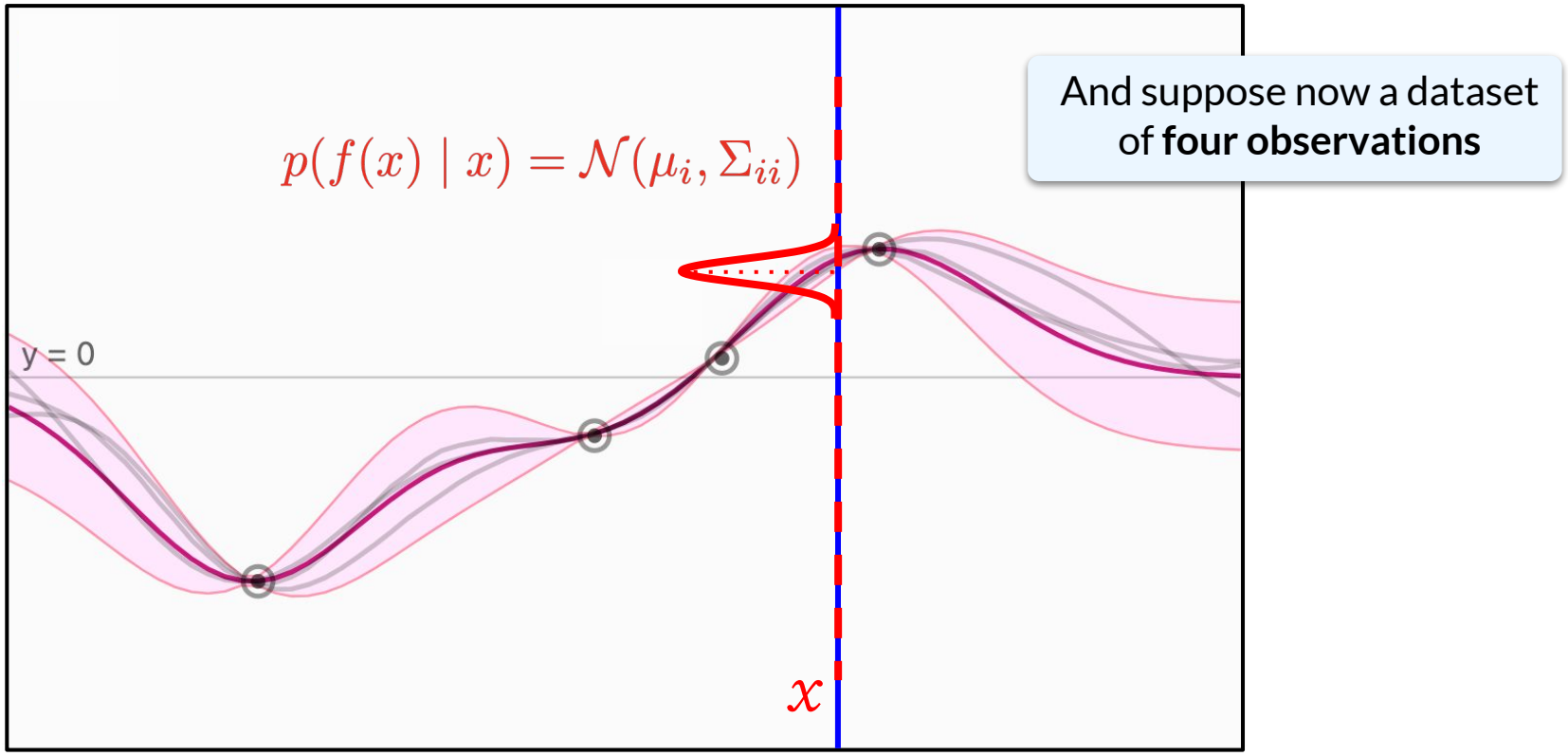
Visualizing this:



Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Posterior

Visualizing this:



Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Summary

GP Summary:

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.
- For any set of input points x, these parameters yield a MVN distribution

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.
- For any set of input points x, these parameters yield a MVN distribution
- And this MVN can be viewed as a **distribution over function values** (for the input points).

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.
- For any set of input points x, these parameters yield a MVN distribution
- And this MVN can be viewed as a **distribution over function values** (for the input points).
- We can then do Bayesian inference:

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.
- For any set of input points x, these parameters yield a MVN distribution
- And this MVN can be viewed as a **distribution over function values** (for the input points).
- We can then do Bayesian inference:
  - Use GP as a prior distribution.
  - Define a Gaussian likelihood for observations.
  - Then compute a GP posterior, given a set of data.

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.
- For any set of input points x, these parameters yield a MVN distribution
- And this MVN can be viewed as a **distribution over function values** (for the input points).
- We can then do Bayesian inference:
  - Use GP as a prior distribution.
  - Define a Gaussian likelihood for observations.
  - Then compute a GP posterior, given a set of data.
- ⇒ This yields a predictive UQ model!

Deep Uncertainty Models

We can also define predictive UQ models using neural networks.

Deep Uncertainty Models

We can also define predictive UQ models using neural networks.

I will go through a few examples of these.

Deep Uncertainty Models

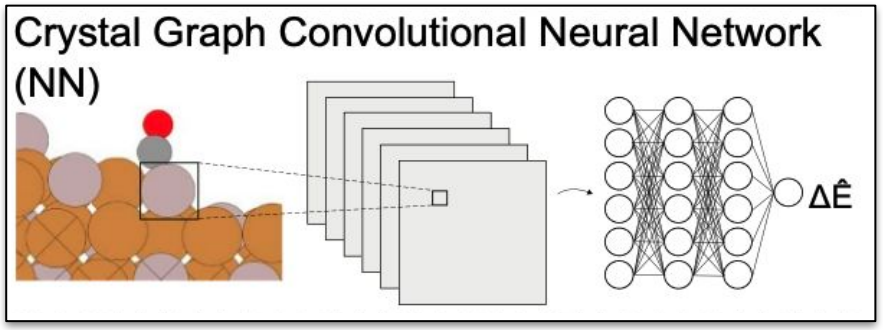
As a running example, we'll look at methods for incorporating predictive uncertainty into the following neural network:

Deep Uncertainty Models —Example: Computational Catalyst Design

As a running example, we'll look at methods for incorporating predictive uncertainty into the following neural network:

Deep Uncertainty Models —Example: Computational Catalyst Design

As a running example, we'll look at methods for incorporating predictive uncertainty into the following neural network:



*Xie, Tian, and Jeffrey C. Grossman. "**Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties."** Physical review letters, 2018.*

CGCNN Model *(a graph convolutional neural network model)*

- *Inputs*: three-dimensional atomic structure (a graph).
- *Outputs*: DFT-calculated site adsorption energies ΔE.

  (⇒ *Regression*).

"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

Suppose we want to *incorporate uncertainty into this neural network*.

(*I.e.*, use neural network for both point predictions and uncertainties).

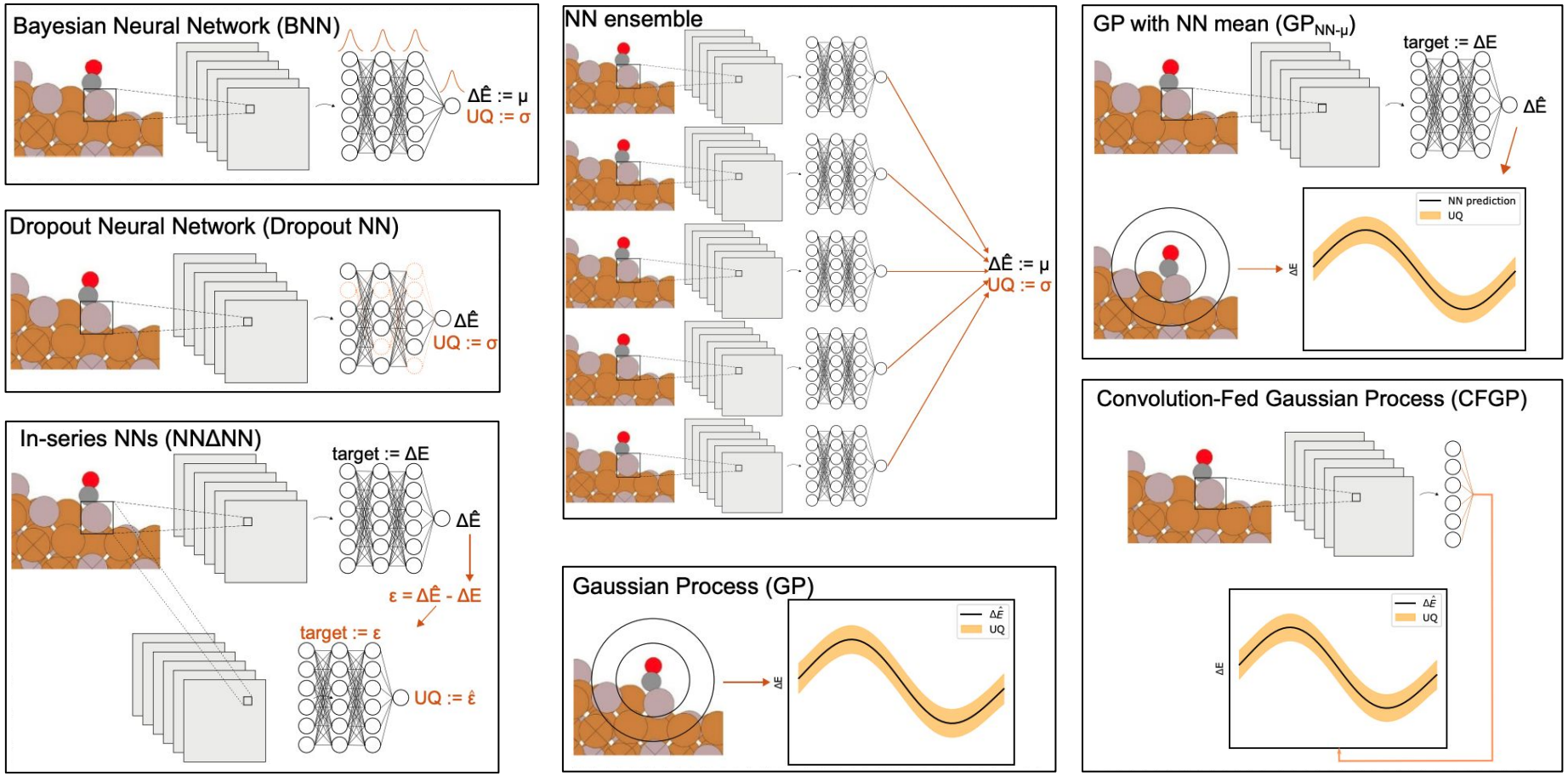
Deep Uncertainty Models —Example: Computational Catalyst Design

Suppose we want to *incorporate uncertainty into this neural network*.

(*I.e.*, use neural network for both point predictions and uncertainties).

⇒ there are many ways this can be done, and it's an active area of research!

Deep Uncertainty Models —Example: Computational Catalyst Design

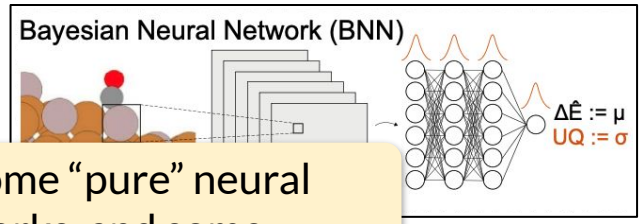


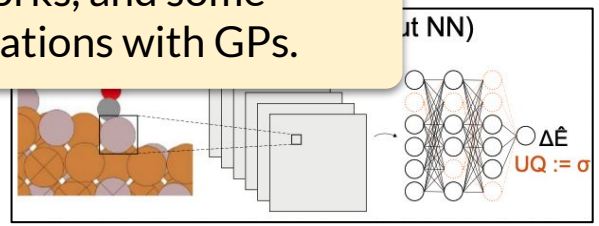
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

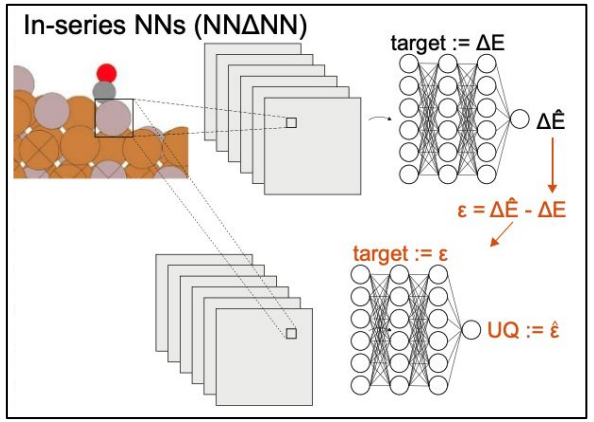
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

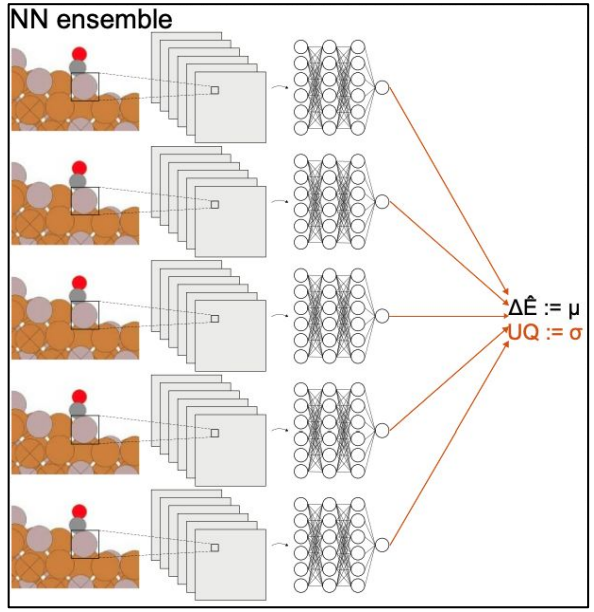
Deep Uncertainty Models —Example: Computational Catalyst Design

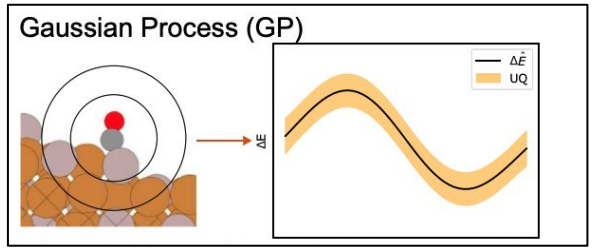
Note: some “pure” neural networks, and some combinations with GPs.

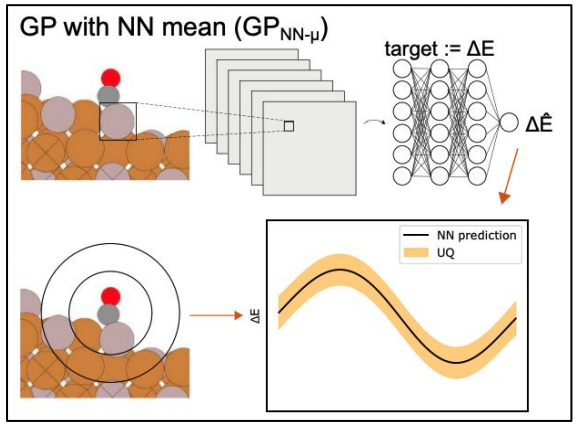


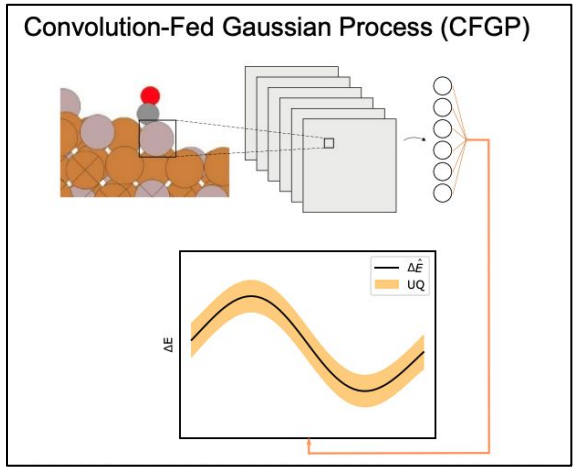








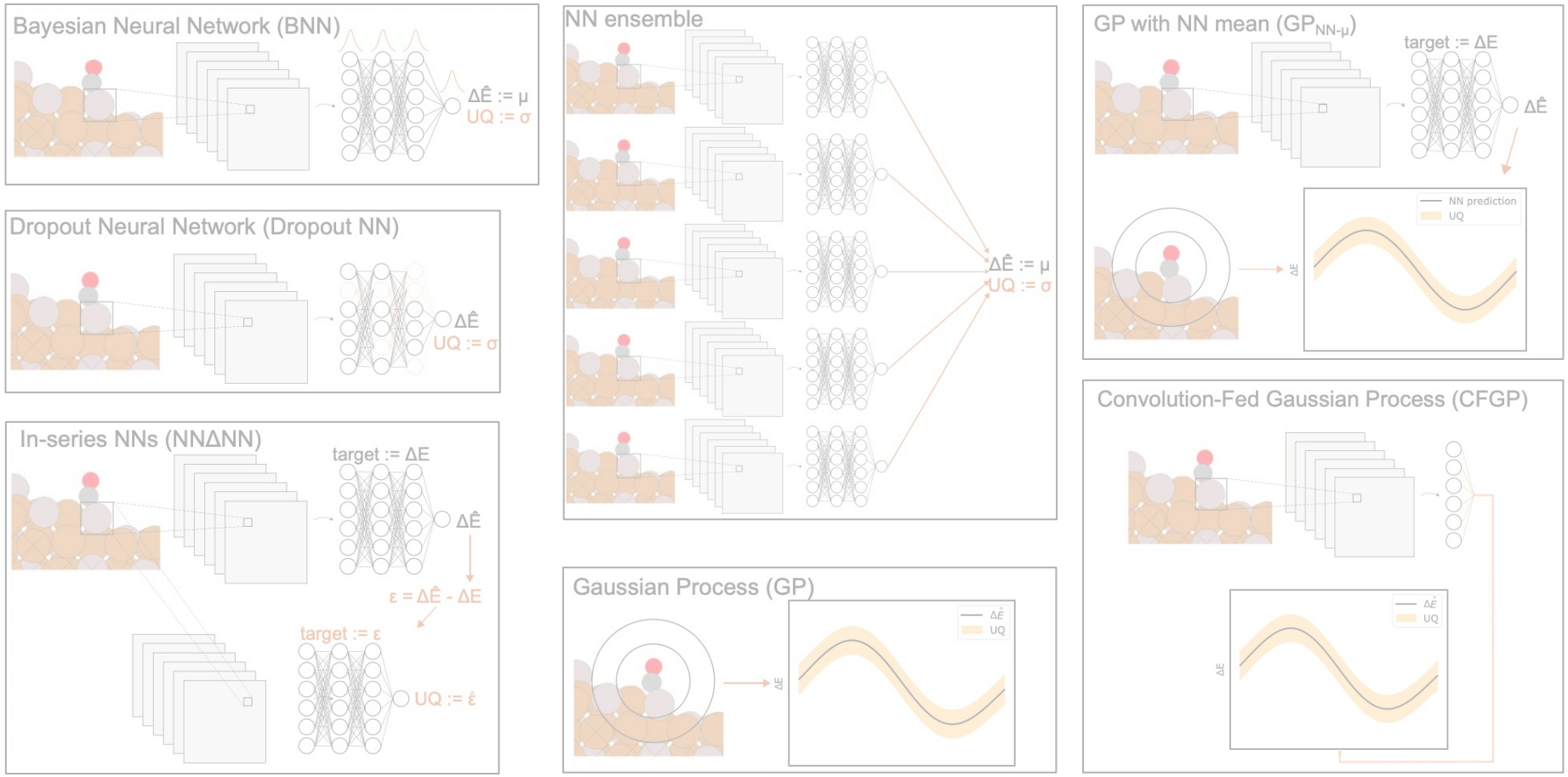




“Methods for comparing uncertainty quantifications for material property predictions”, *Tran, *Neiswanger, et al., MLST. 2020

“Computational catalyst discovery: Active classification through myopic multiscale sampling”, *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

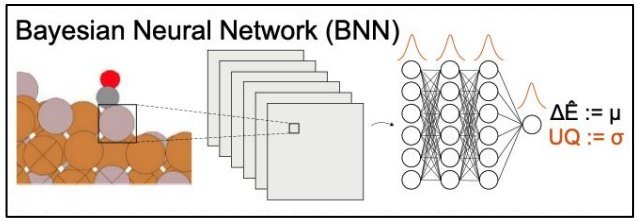
Deep Uncertainty Models —Example: Computational Catalyst Design

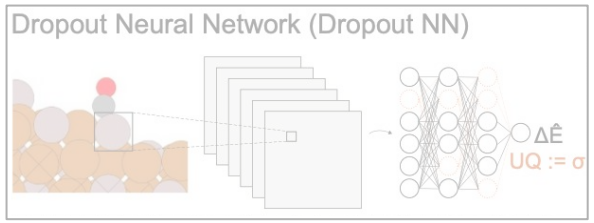


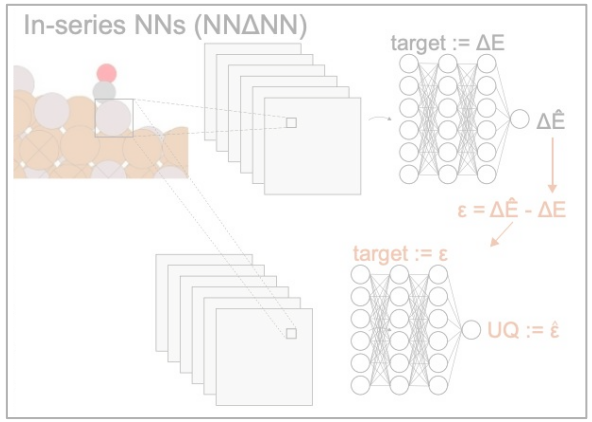
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

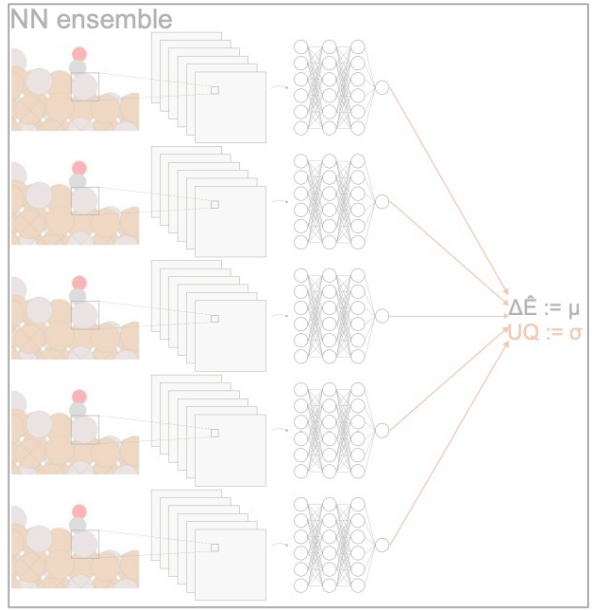
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

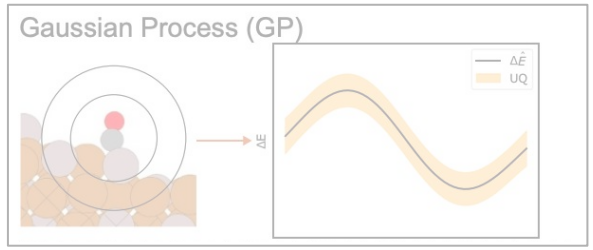
Deep Uncertainty Models —Example: Computational Catalyst Design

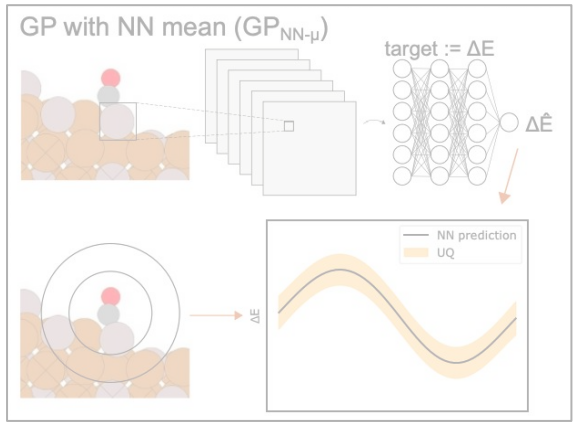


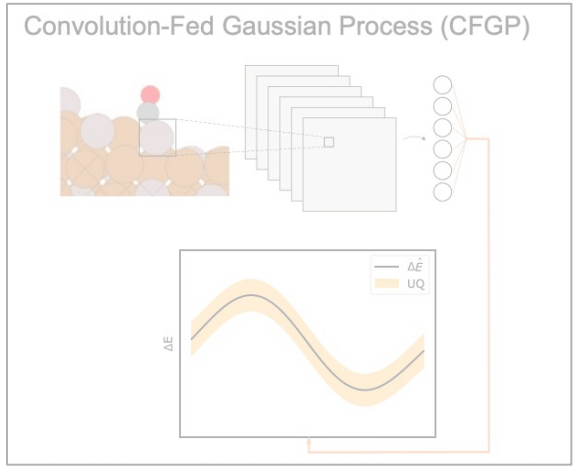








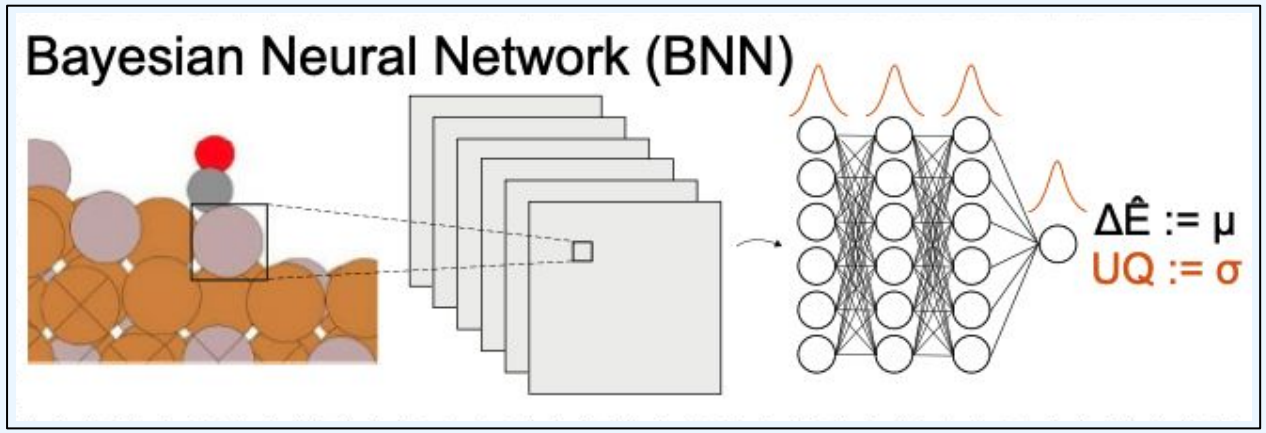




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

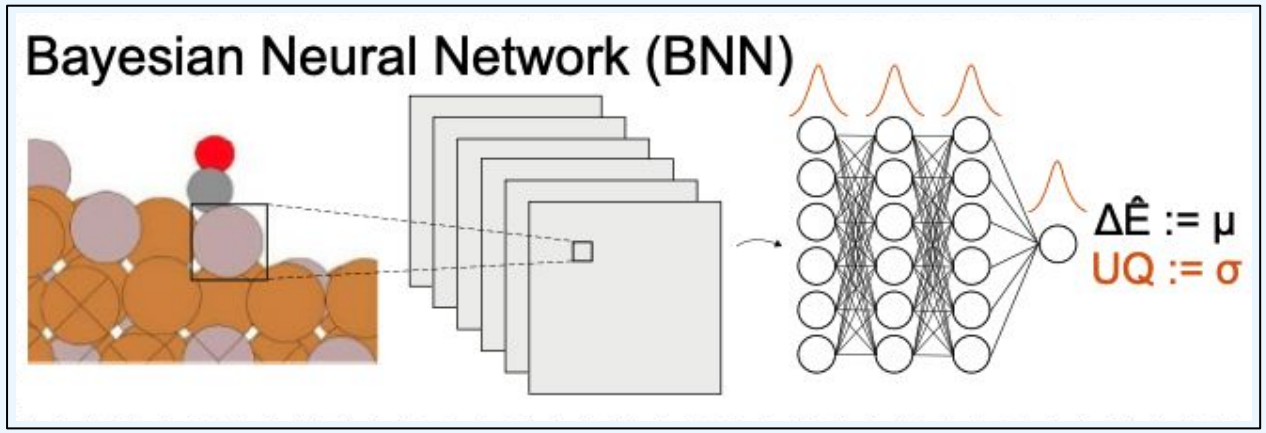
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

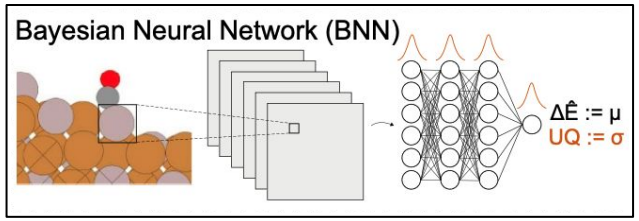


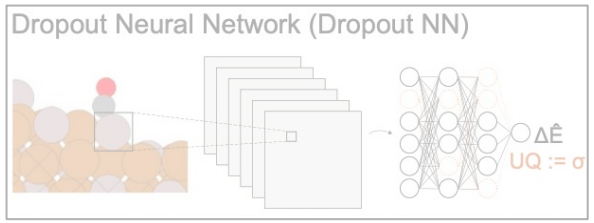
- Define a prior on neural network weights.
- In training, compute posterior over neural network weights (rather than a point estimate).
- Often using some type of MCMC (*e.g.*, LMC) or variational inference.

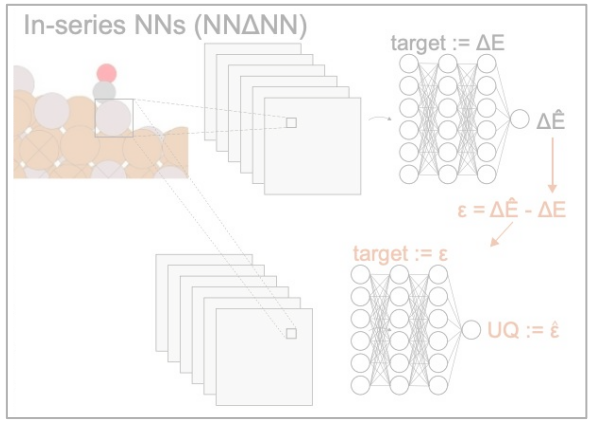
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

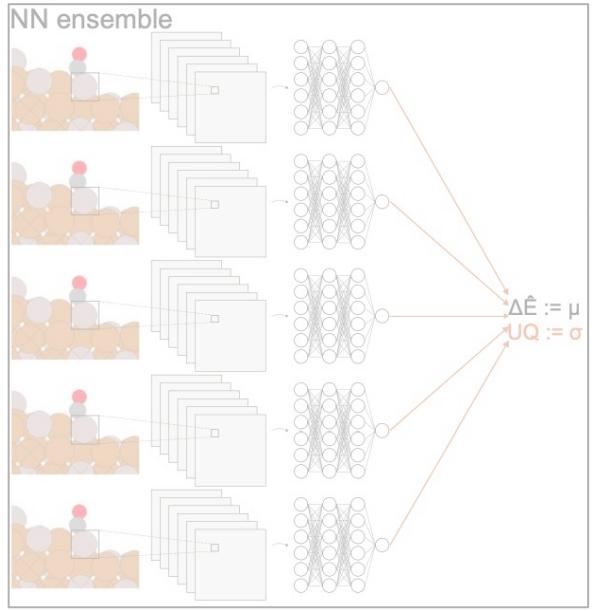
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

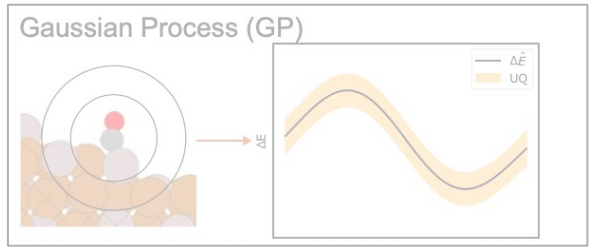
Deep Uncertainty Models —Example: Computational Catalyst Design

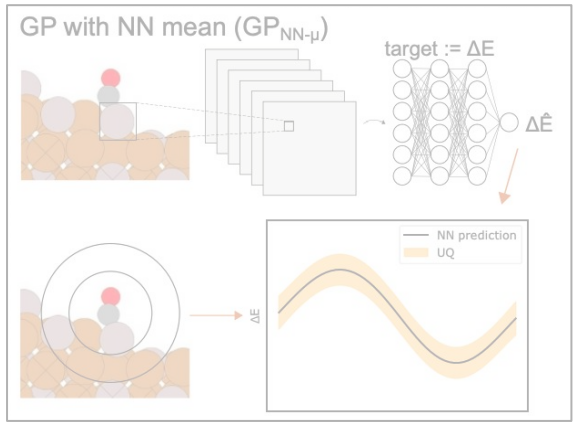


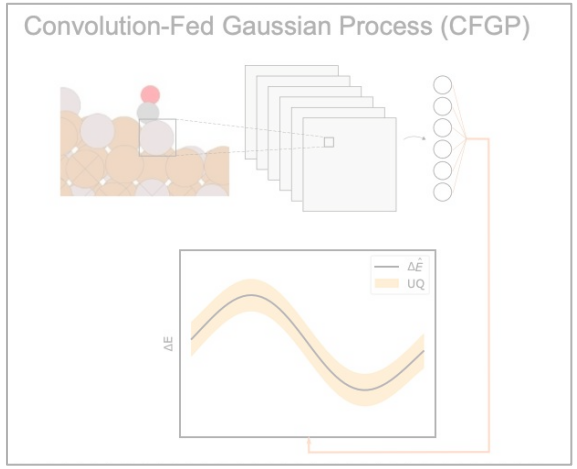








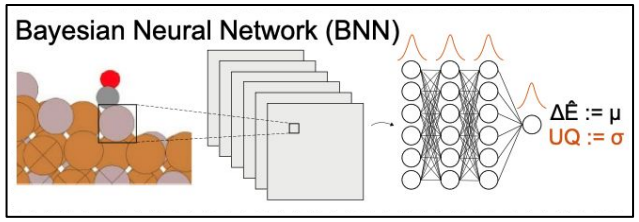


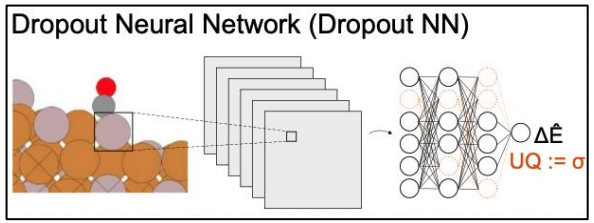


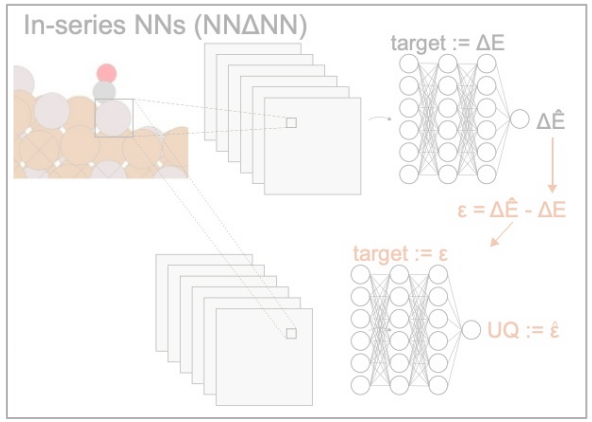
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

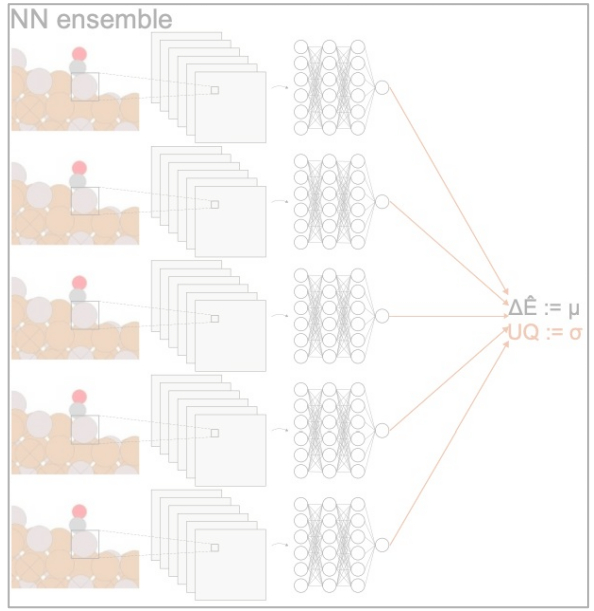
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

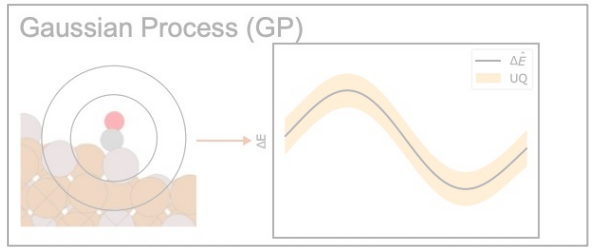
Deep Uncertainty Models —Example: Computational Catalyst Design

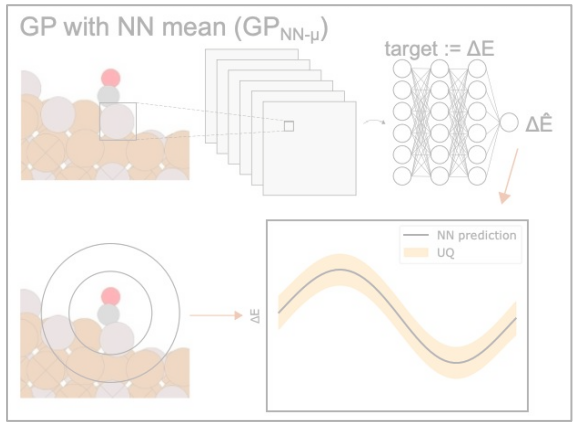


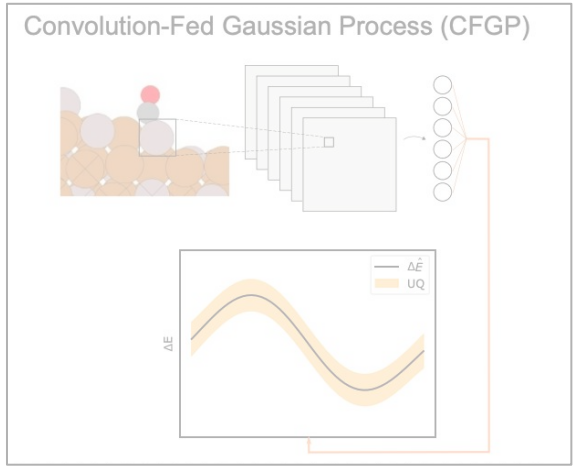








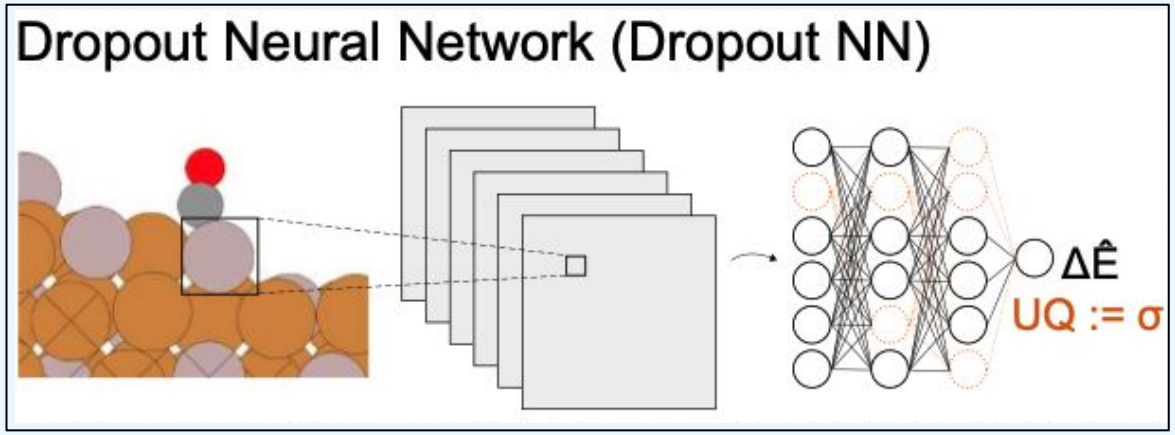




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

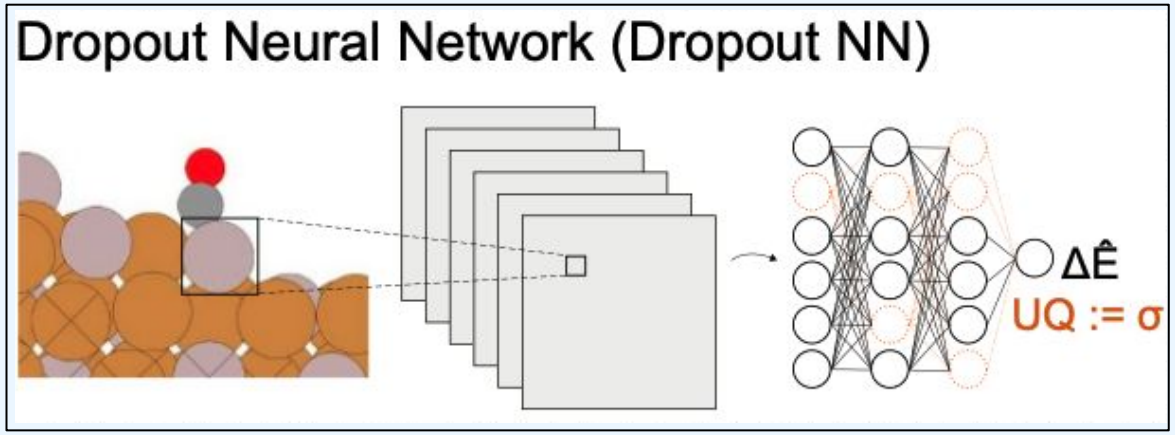
Deep Uncertainty Models —Example: Computational Catalyst Design



“Methods for comparing uncertainty quantifications for material property predictions”, *Tran, *Neiswanger, et al., MLST. 2020

“Computational catalyst discovery: Active classification through myopic multiscale sampling”, *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

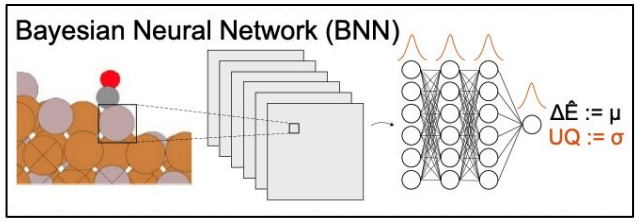


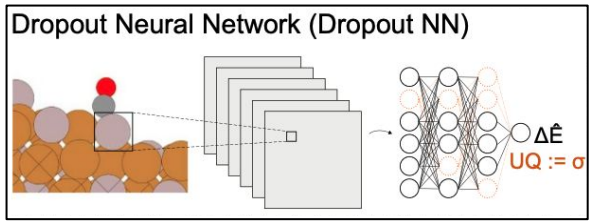
- During train/inference-time, perform *Monte Carlo Dropout*.
- (*I.e.*, set a random fraction of the neuron's outputs to zero during forward pass of the model).
- ⇒ A set of forward passes yields a set of samples from an output distribution.

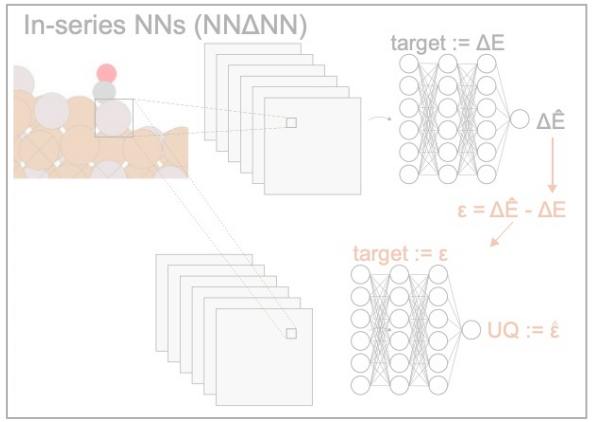
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

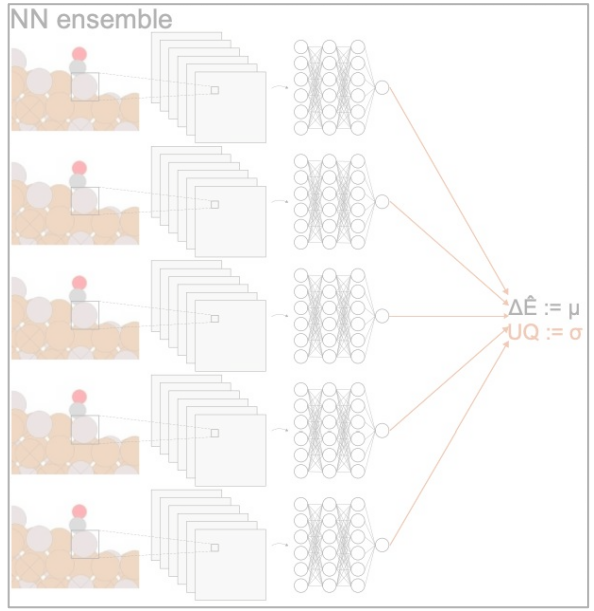
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

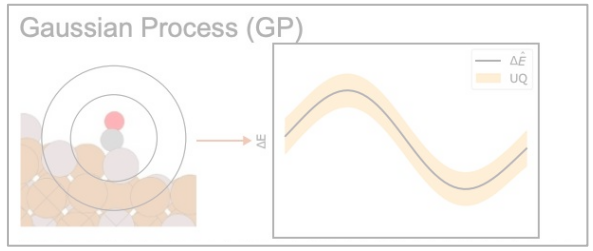
Deep Uncertainty Models —Example: Computational Catalyst Design

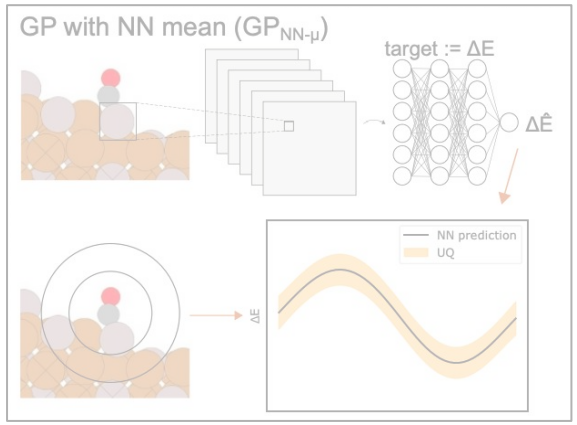


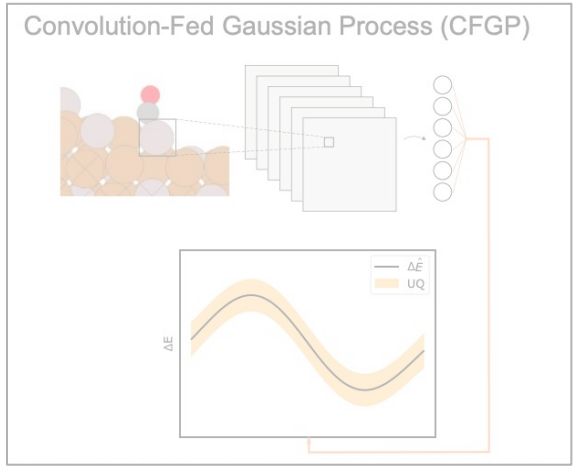








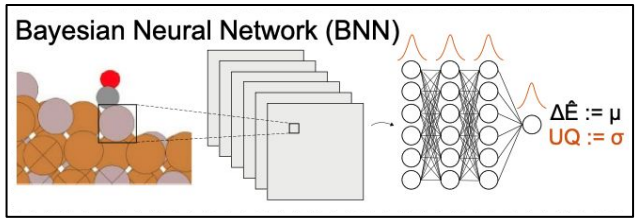


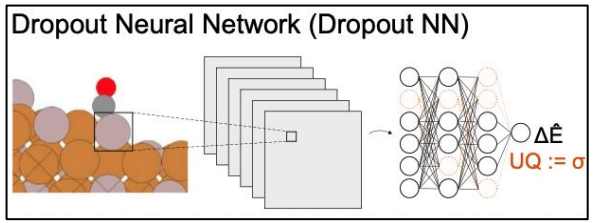


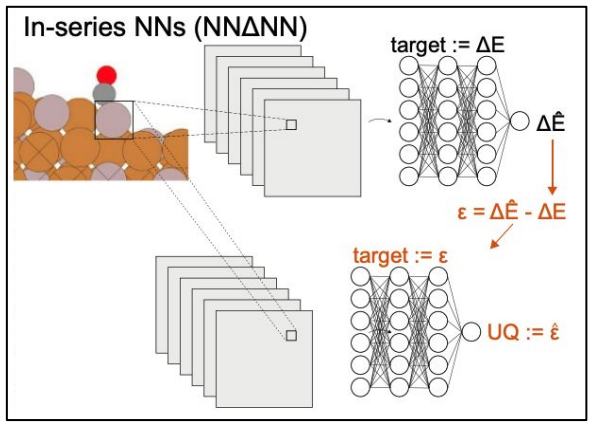
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

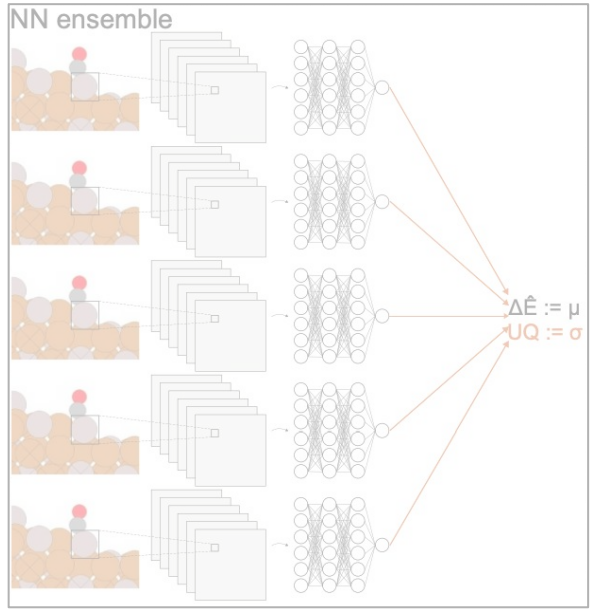
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

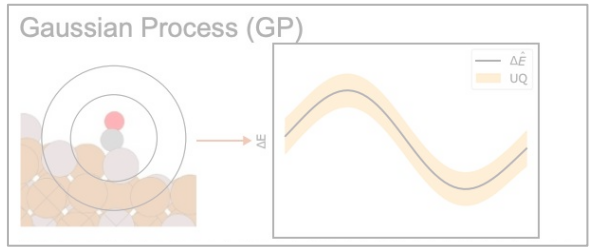
Deep Uncertainty Models —Example: Computational Catalyst Design

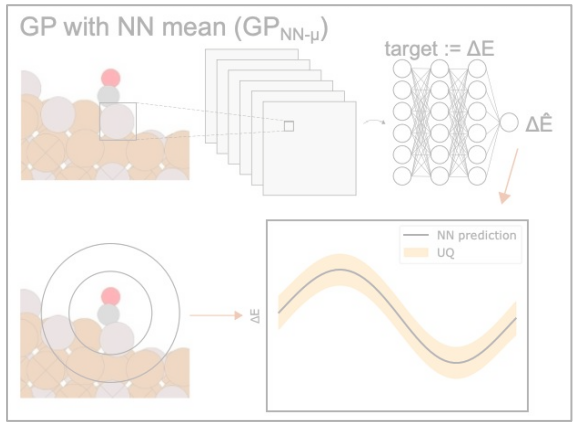


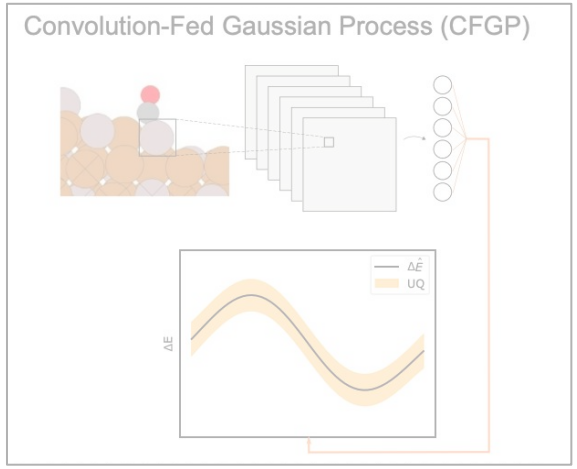








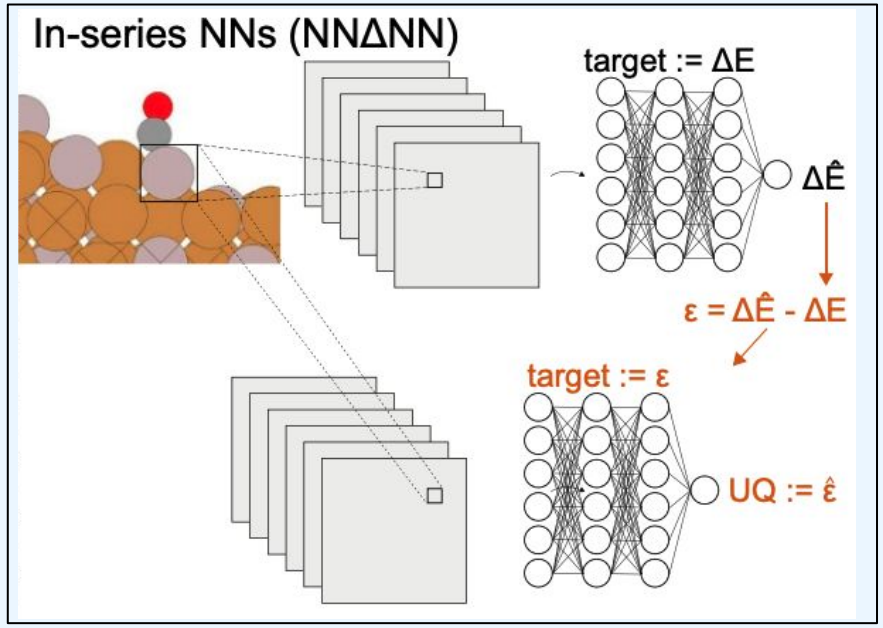




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

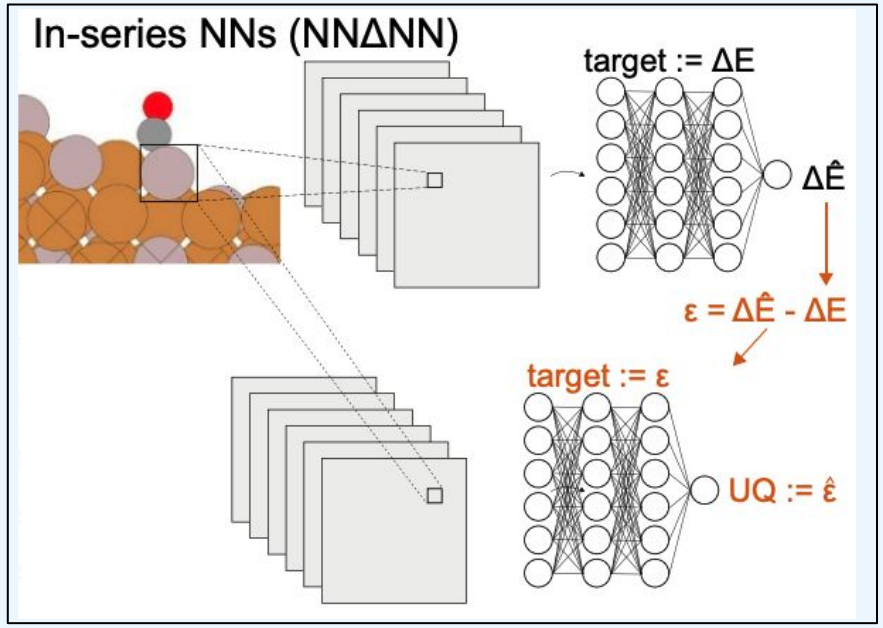
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design



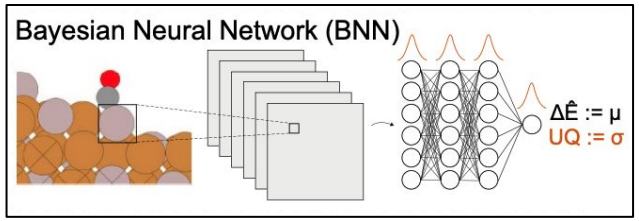
- First train a neural network for point prediction.
- Then compute residuals on a held-out set.
- Then use another neural network to try and predict residuals.

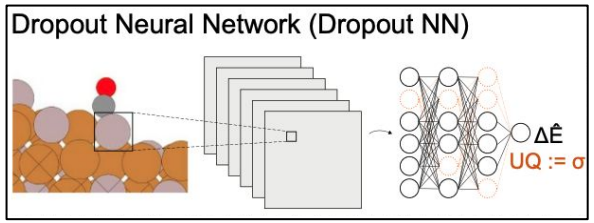
- ⇒ Second network predicts error, which can be used as UQ model.

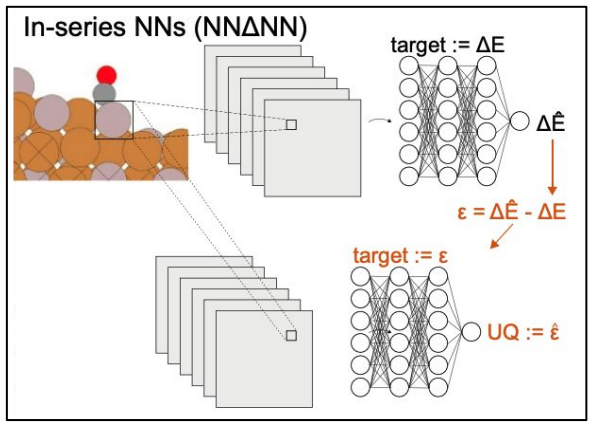
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

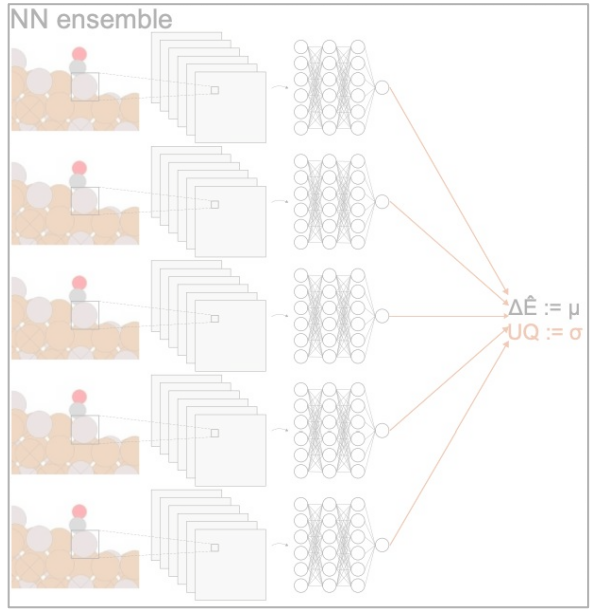
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

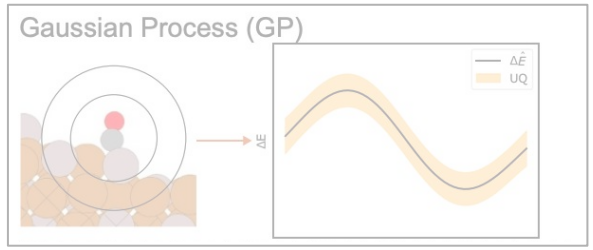
Deep Uncertainty Models —Example: Computational Catalyst Design

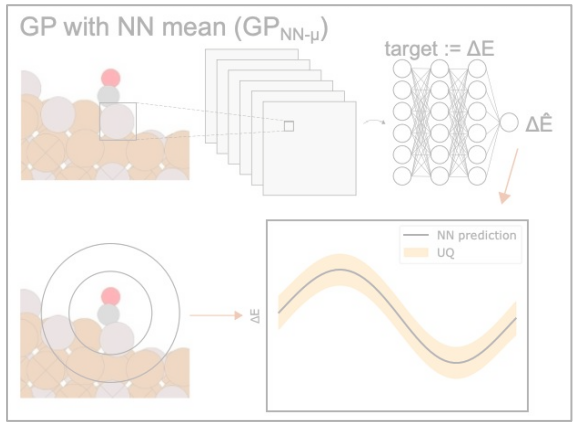


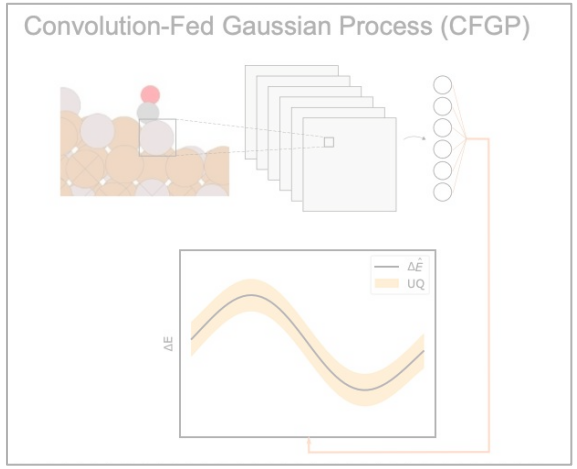








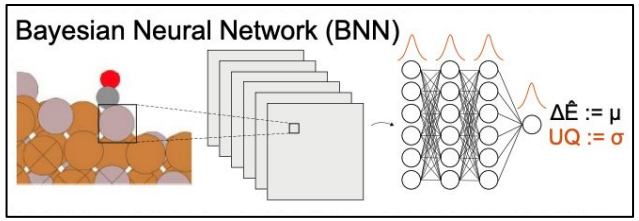


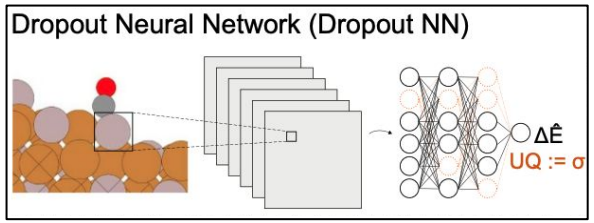


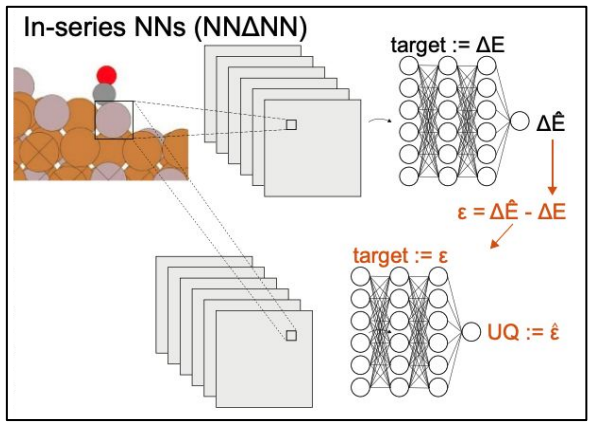
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

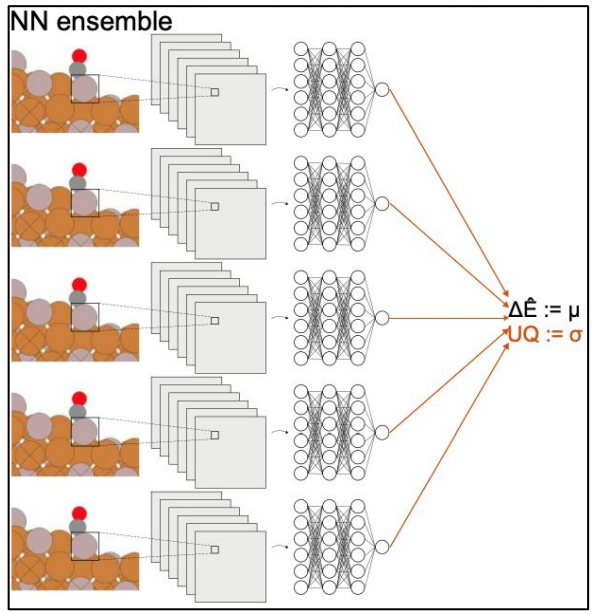
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

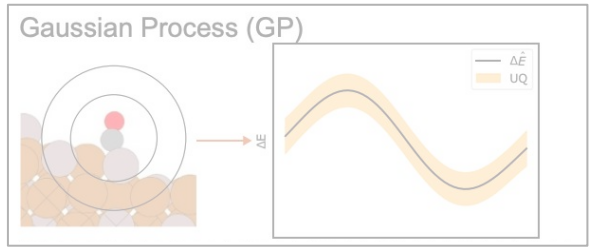
Deep Uncertainty Models —Example: Computational Catalyst Design

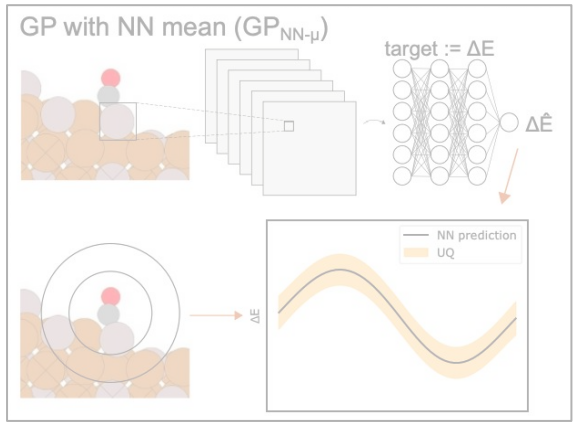


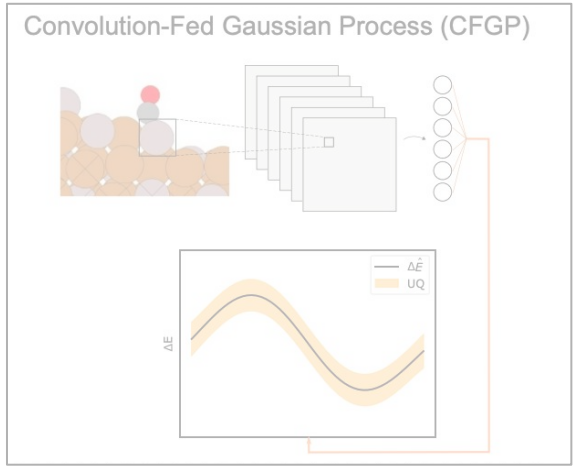








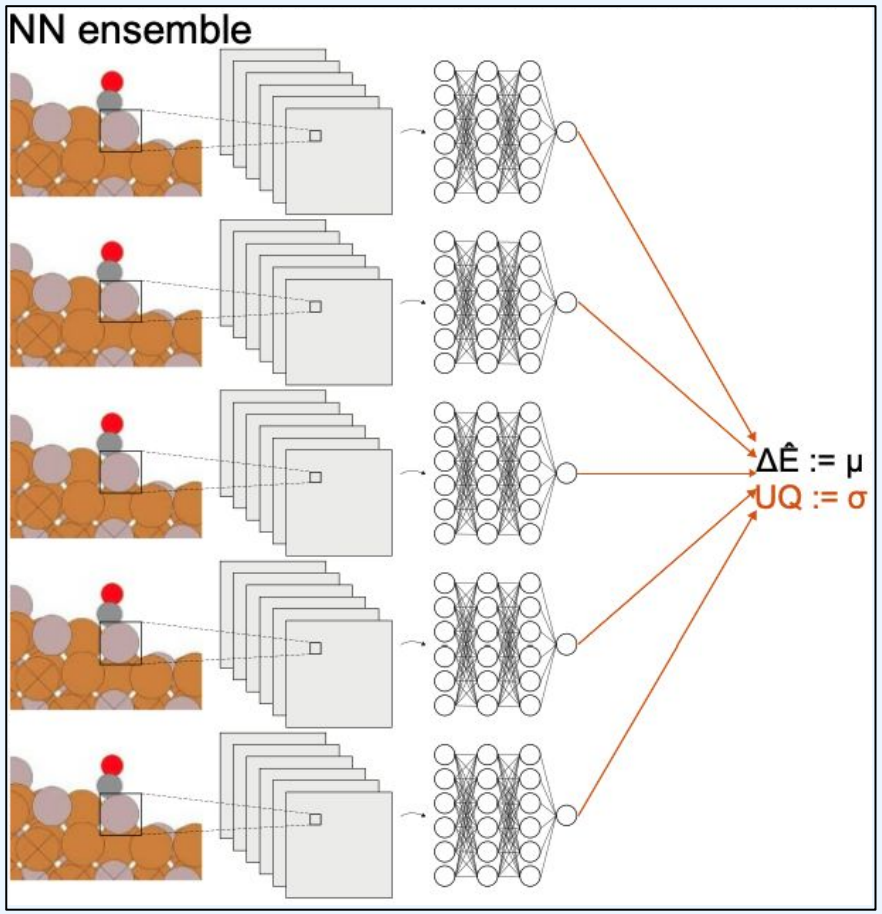




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

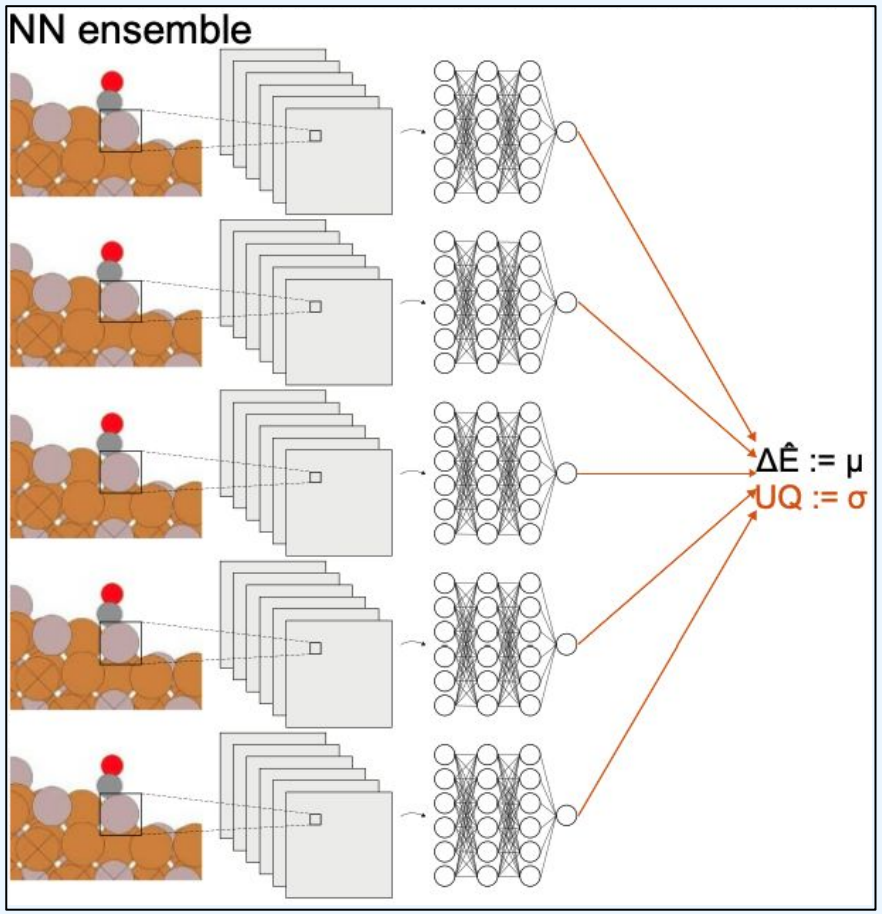
Deep Uncertainty Models —Example: Computational Catalyst Design



“Methods for comparing uncertainty quantifications for material property predictions”, *Tran, *Neiswanger, et al., MLST. 2020

“Computational catalyst discovery: Active classification through myopic multiscale sampling”, *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

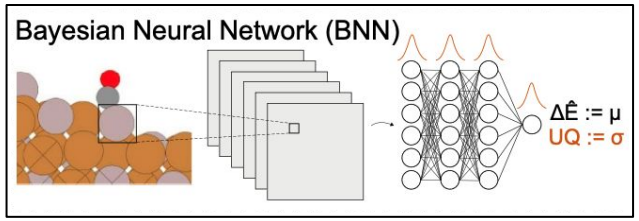


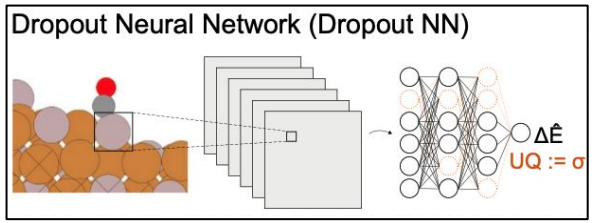
- Train a set of neural networks.
- (Often each neural network has a different initialization, subset of data, or hyperparameter setting).
- Use diversity of predictions (agreement) as a measure of uncertainty in predictions.

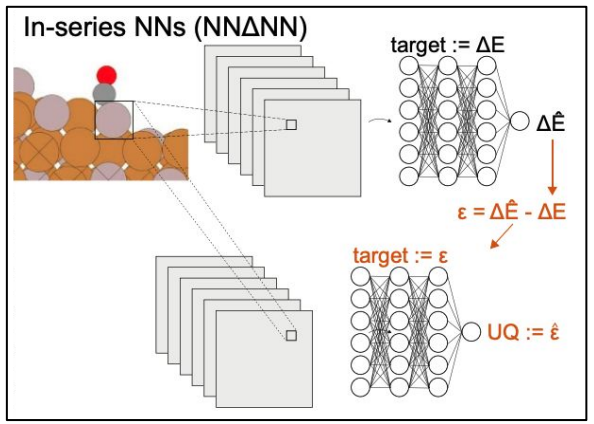
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

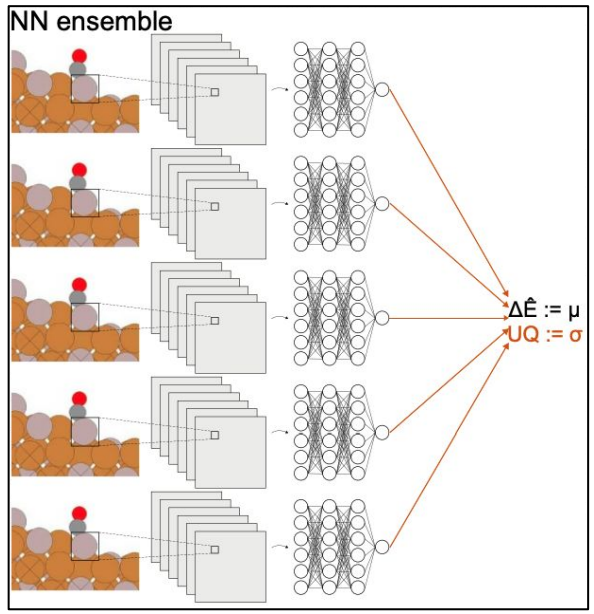
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

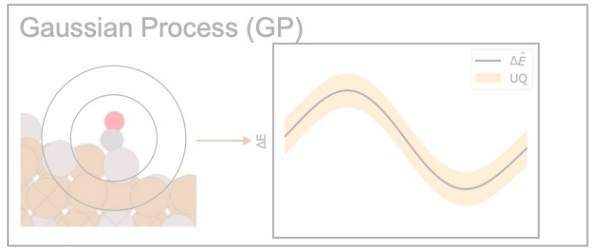
Deep Uncertainty Models —Example: Computational Catalyst Design

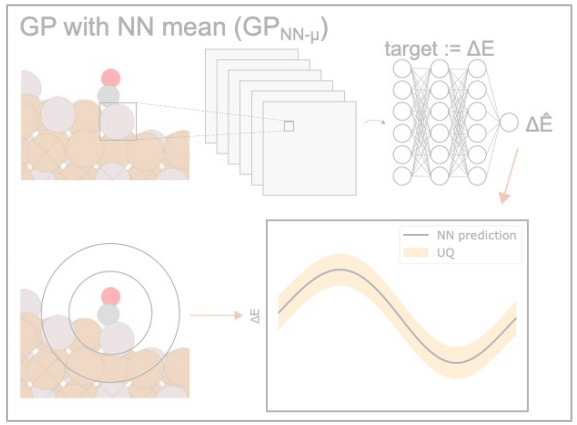


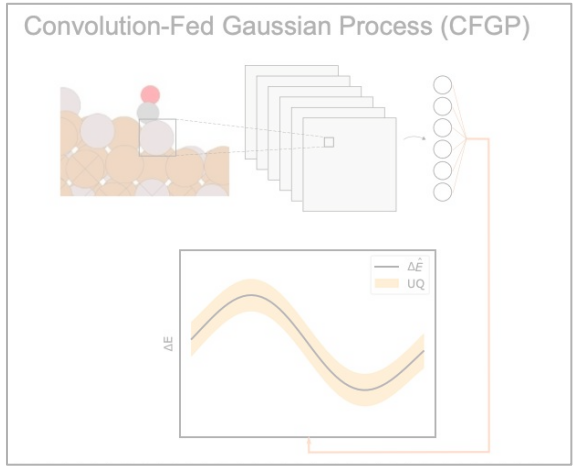








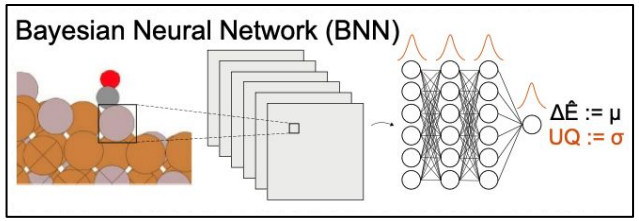


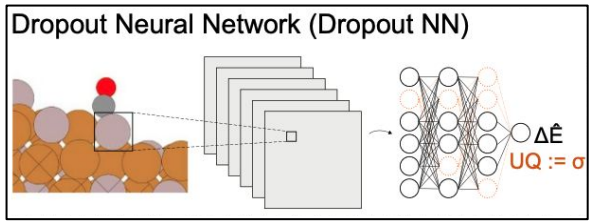


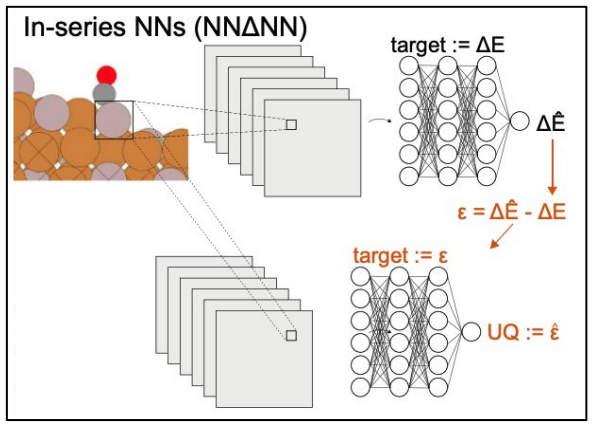
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

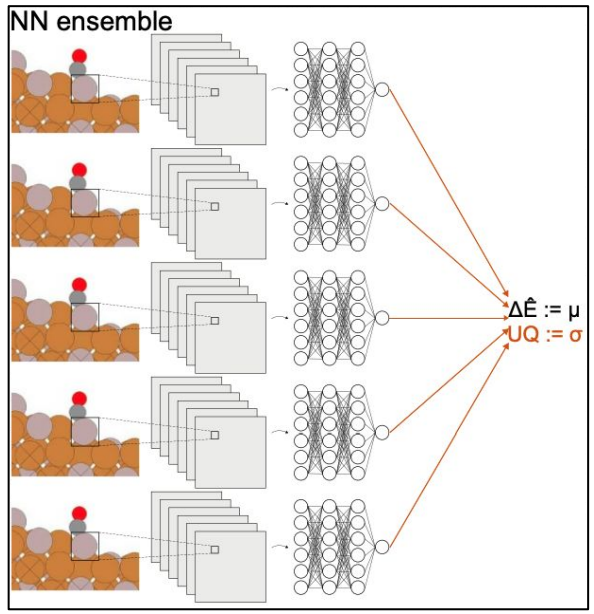
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

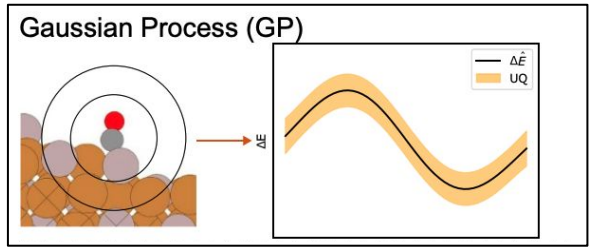
Deep Uncertainty Models —Example: Computational Catalyst Design

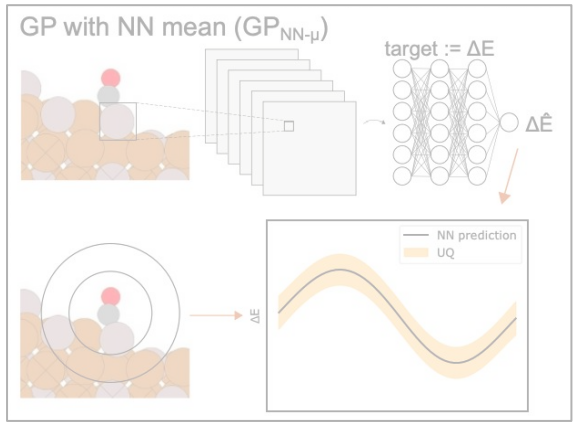


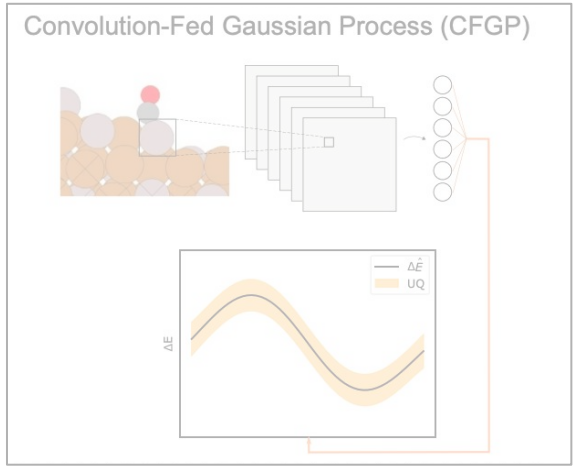








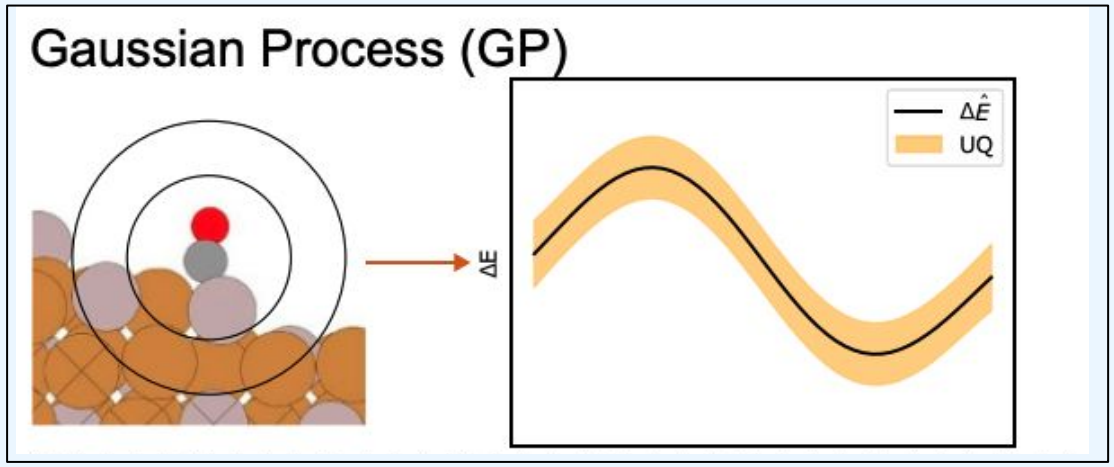




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

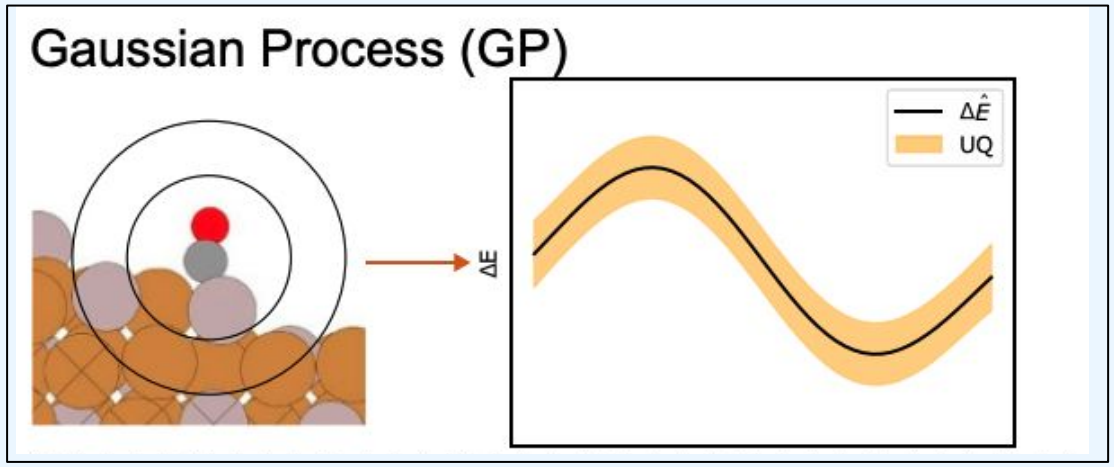
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

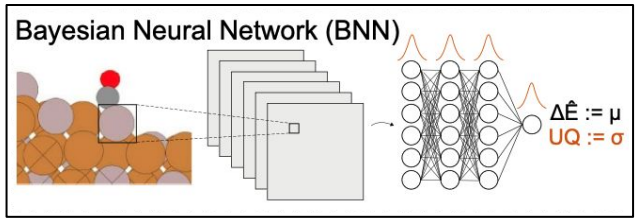


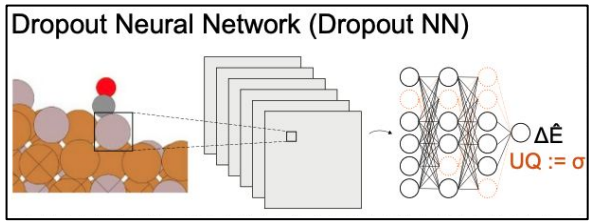
- Define some appropriate mean and kernel function on graph inputs.
- This yields a valid GP, which can make predictions on new inputs (new graphs)
- *Difficulty*: hard to hand-design good kernel function, & doesn't use NN.

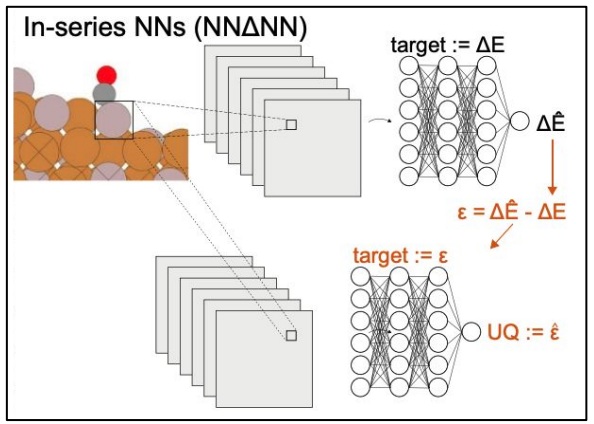
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

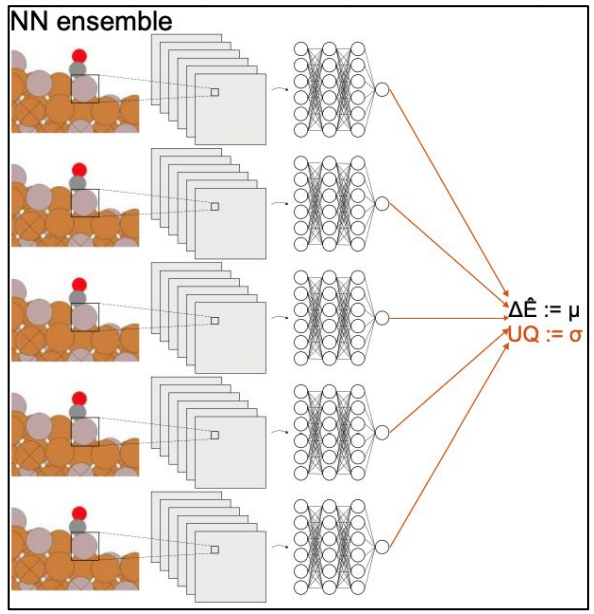
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

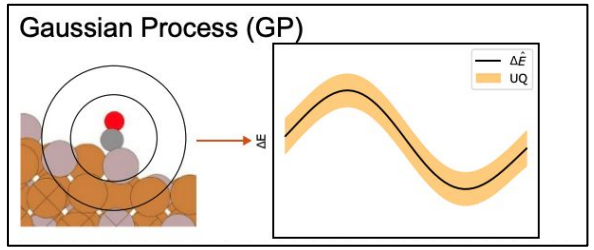
Deep Uncertainty Models —Example: Computational Catalyst Design

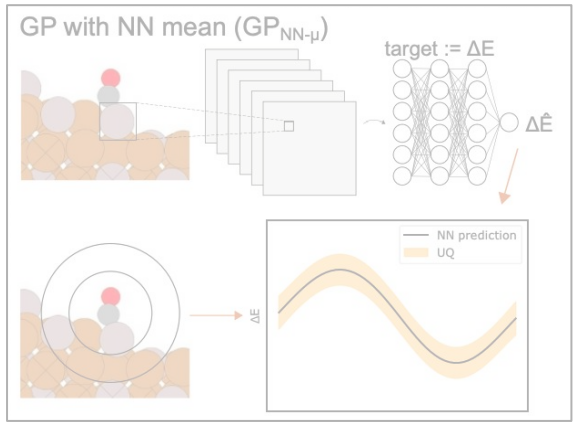


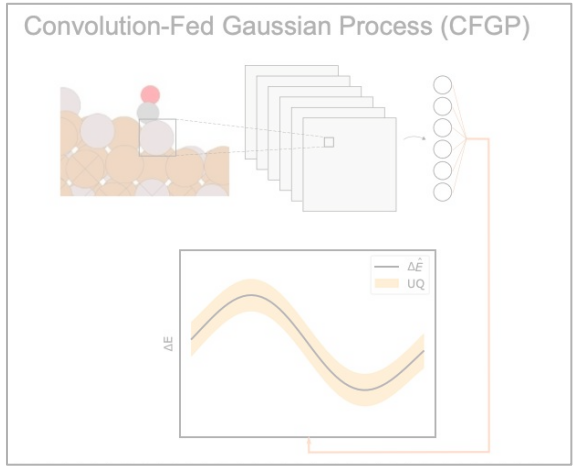








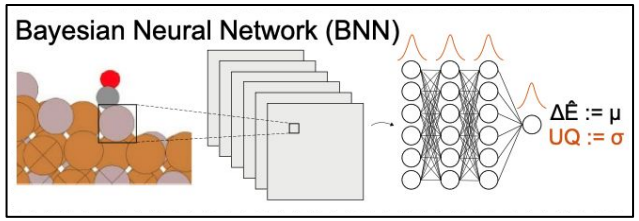


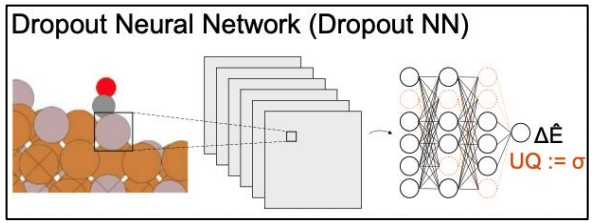


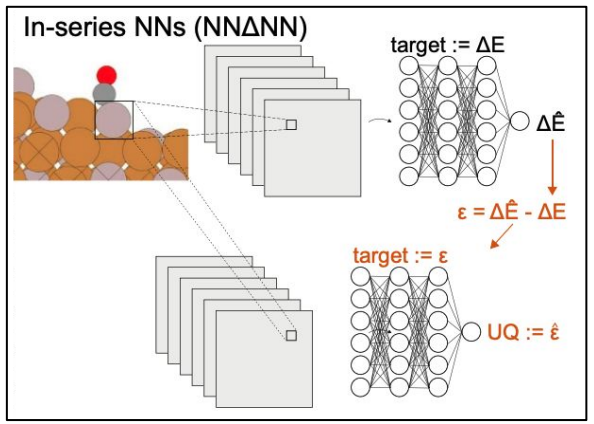
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

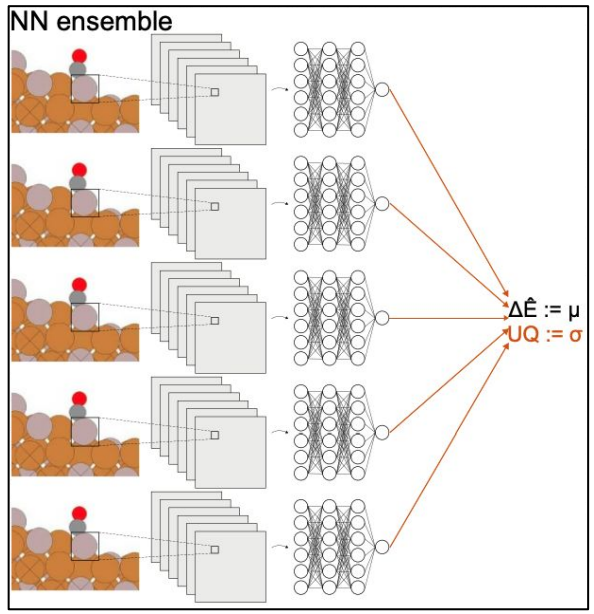
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

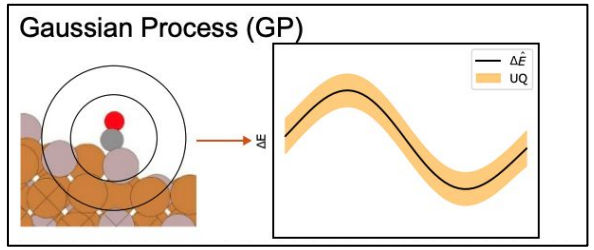
Deep Uncertainty Models —Example: Computational Catalyst Design

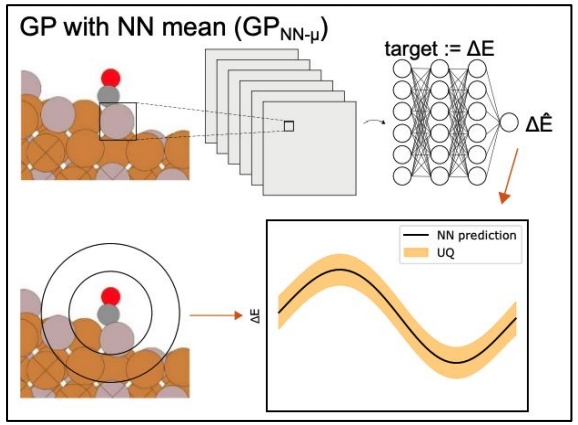


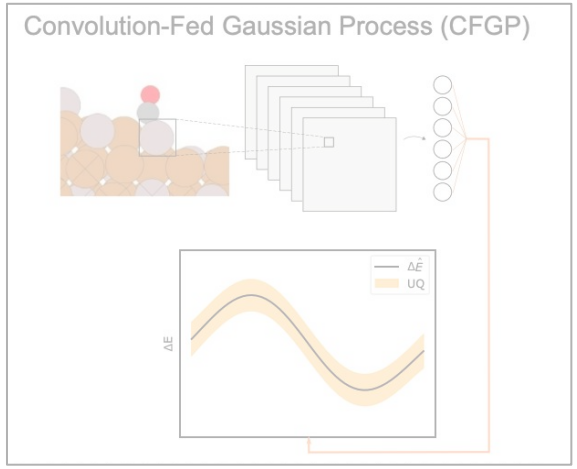








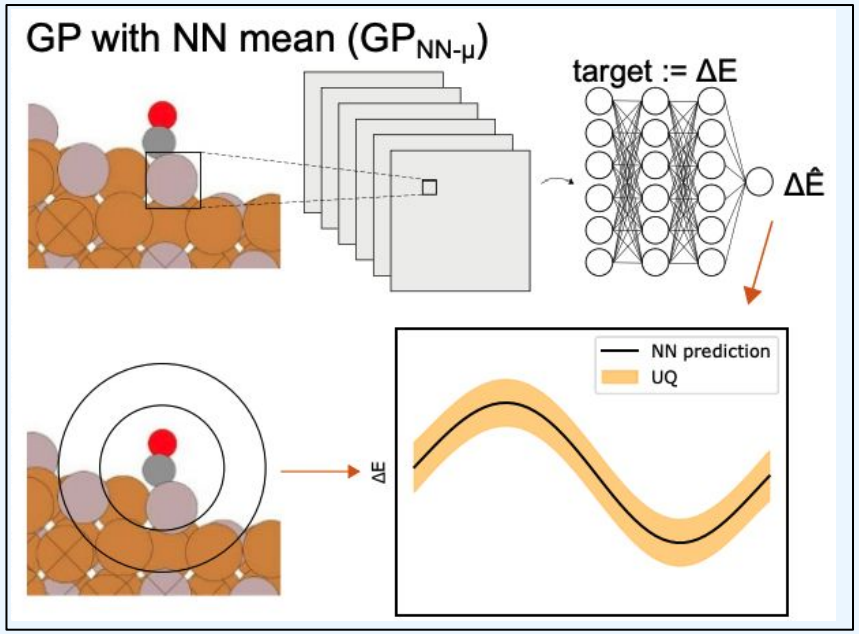




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

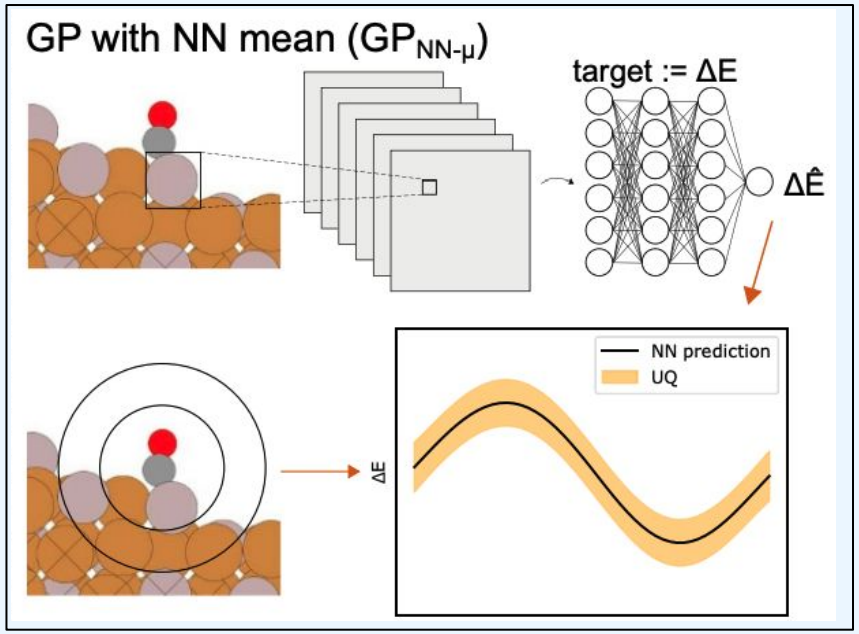
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design



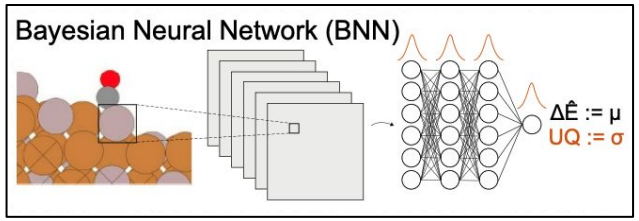
- In GPs, the prior mean fn is usually set to be **0.**
- But it could be anything…
- Here, we can set the prior mean function to be the output of a neural network model!

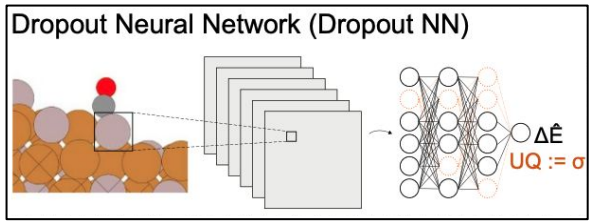
- Downside: still need to define a kernel function on graph inputs.

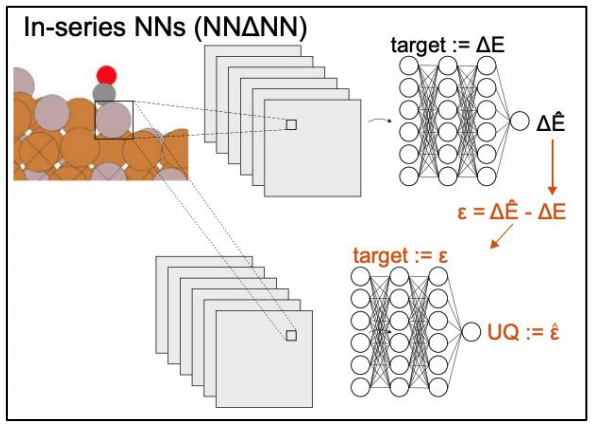
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

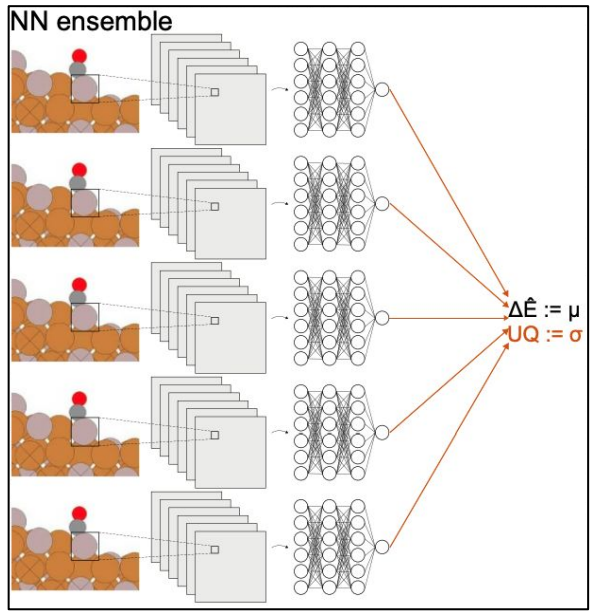
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

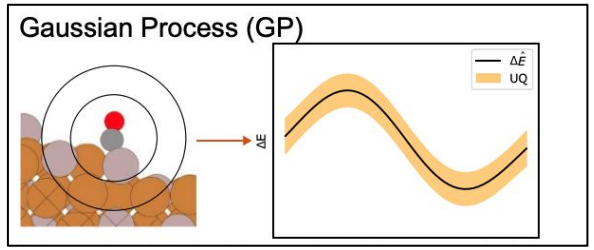
Deep Uncertainty Models —Example: Computational Catalyst Design

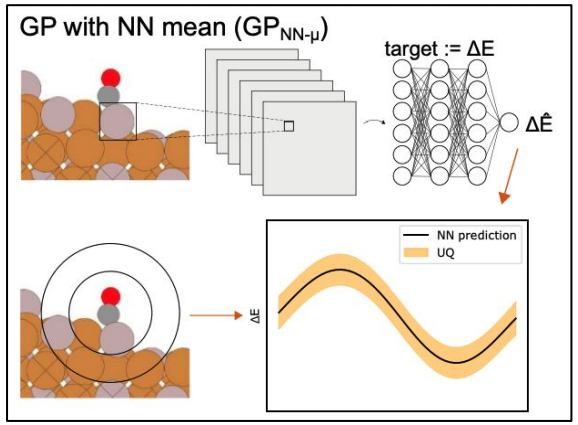


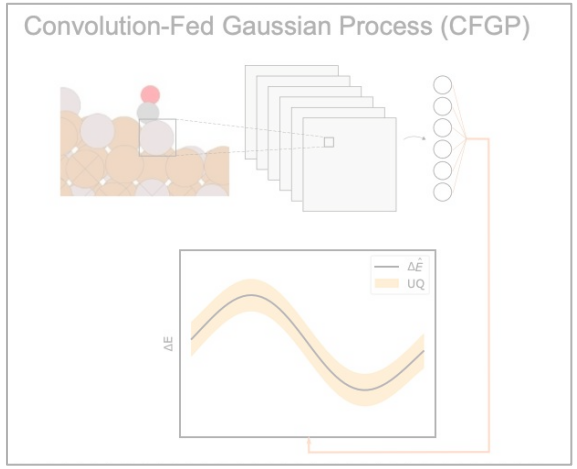








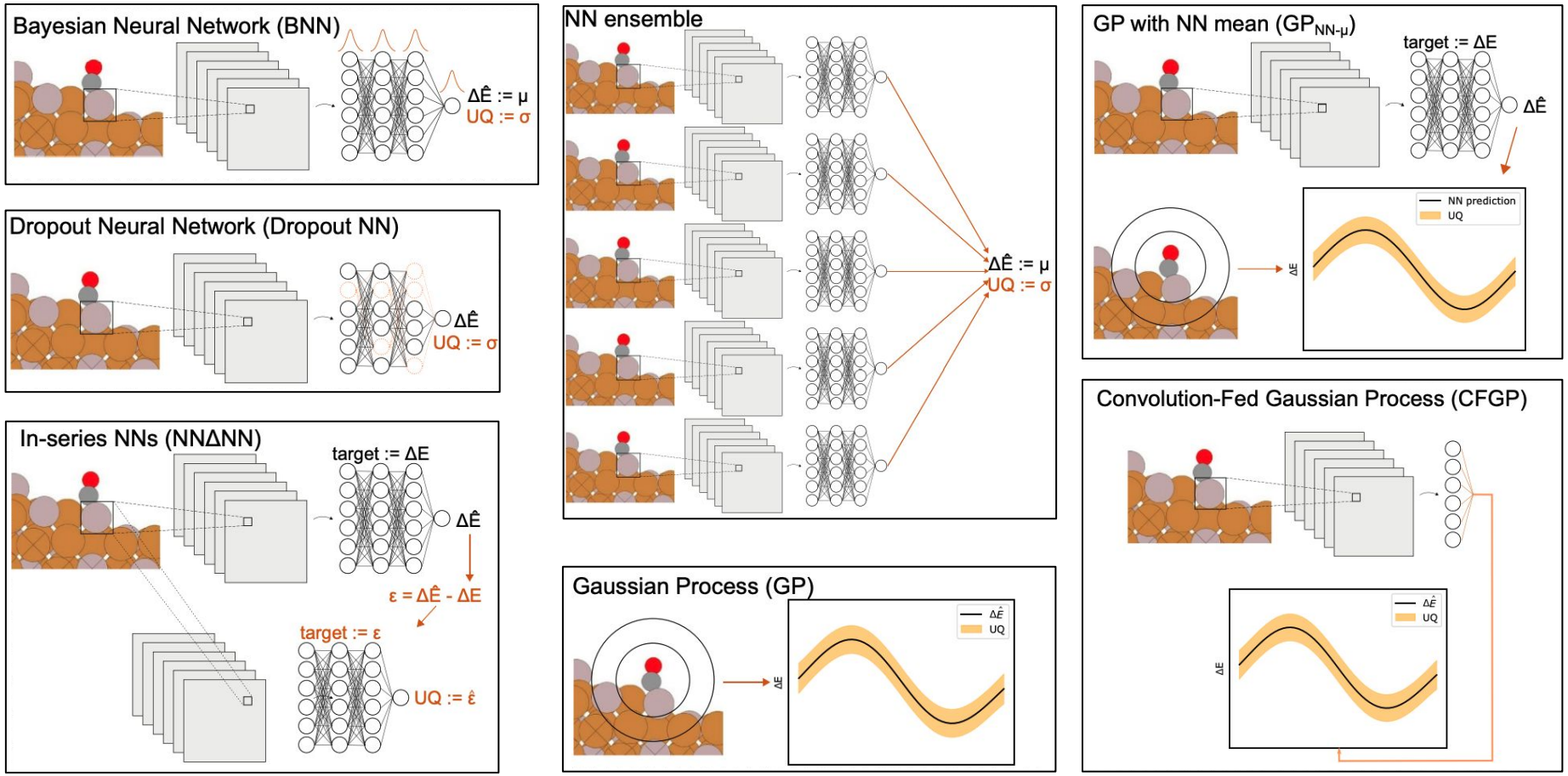




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

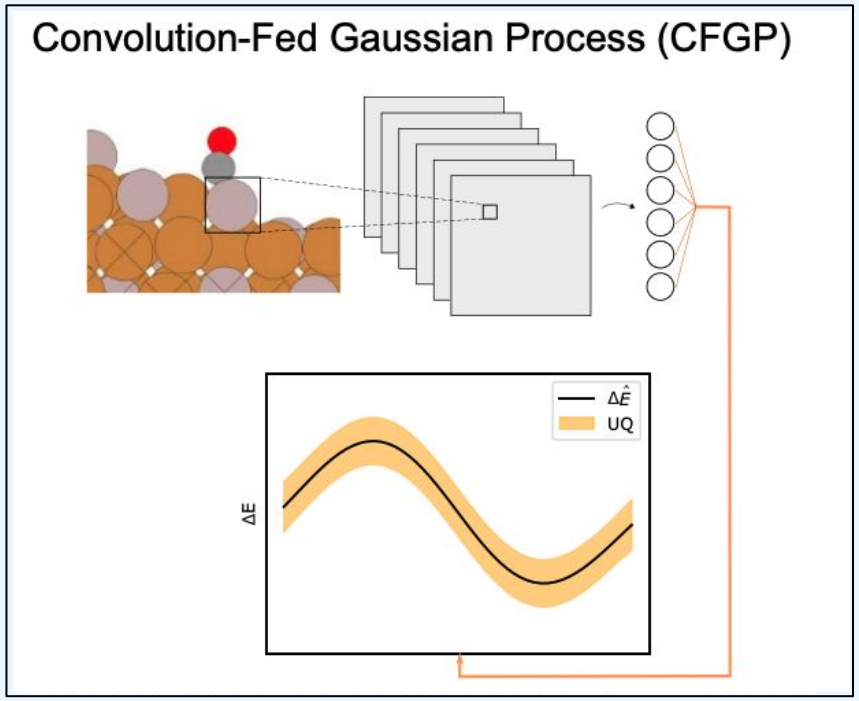
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

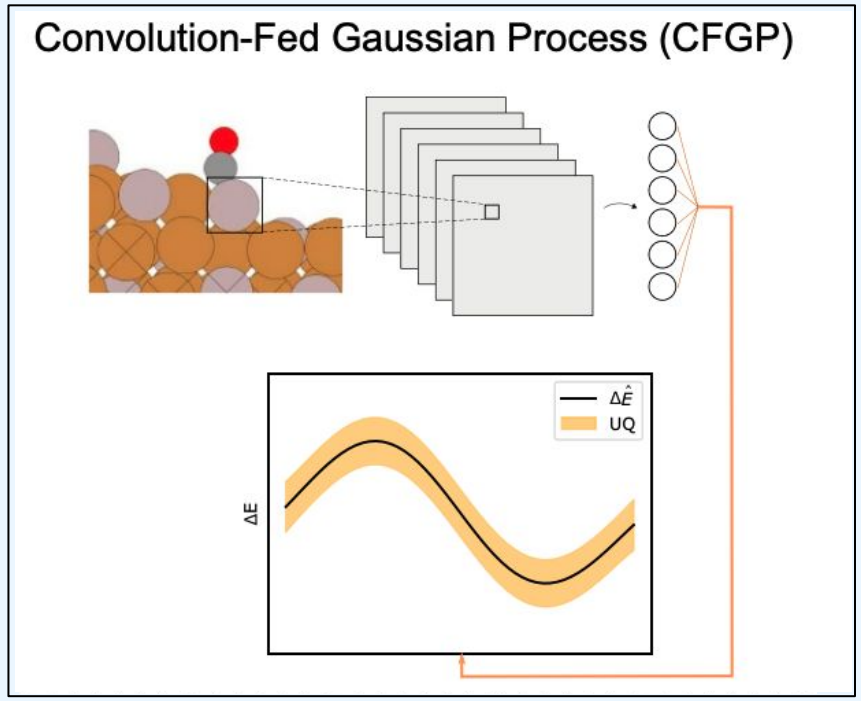
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

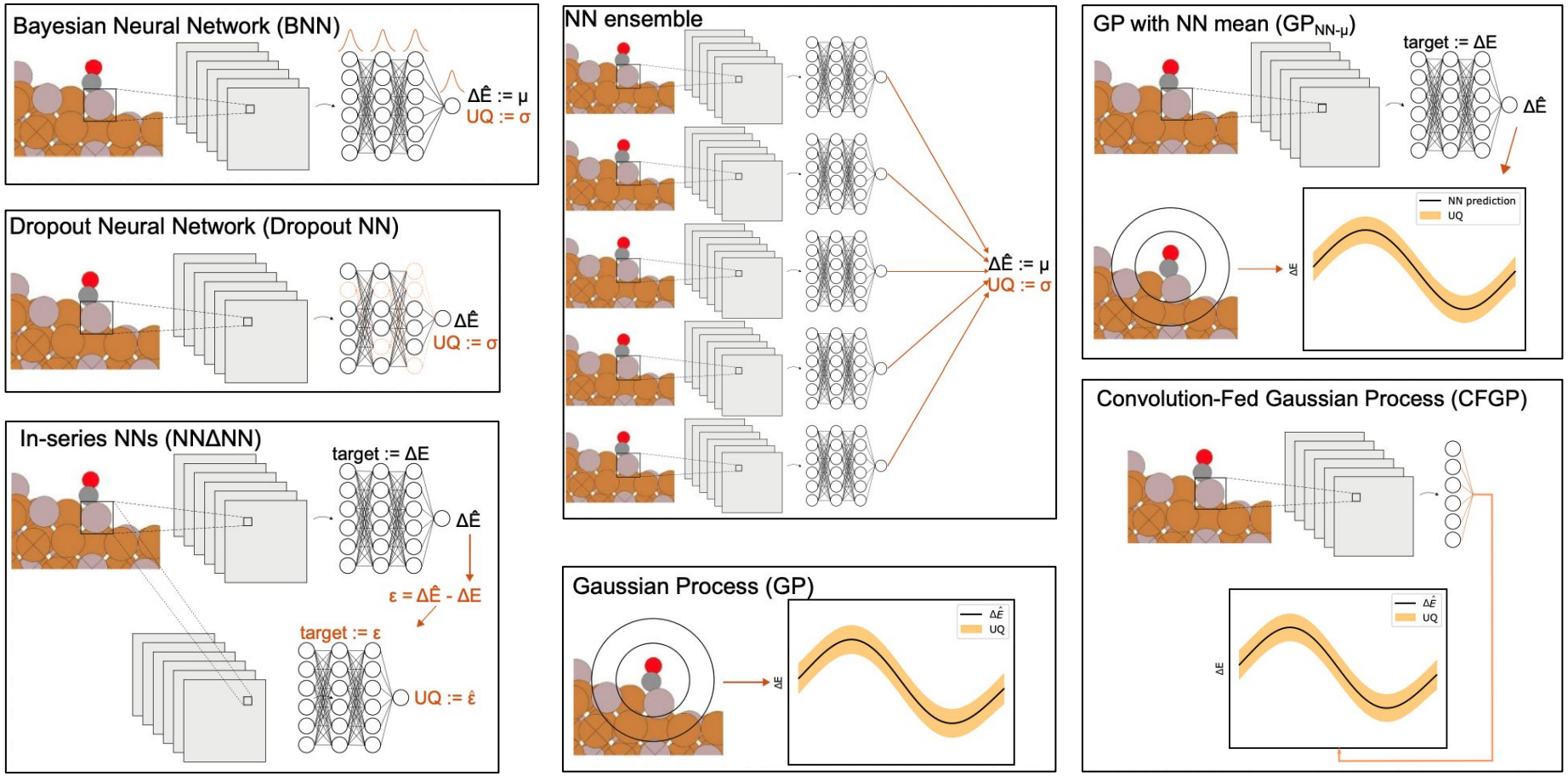


- Here, we use the neural network to learn good representations.
- For any input, first make a prediction w/ neural network.
- Use output at an intermediate layer as new transformed input (into a continuous space).
- Then define a Gaussian process on this new representation of inputs.

"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Next Time

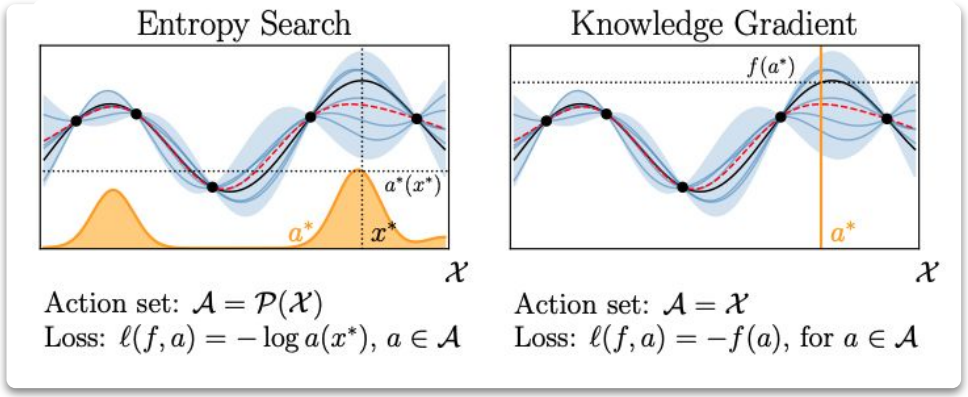
Next Time

Lecture: Using predictive UQ models for active learning and sequential decision making, including Bayesian optimization and optimal experimental design.

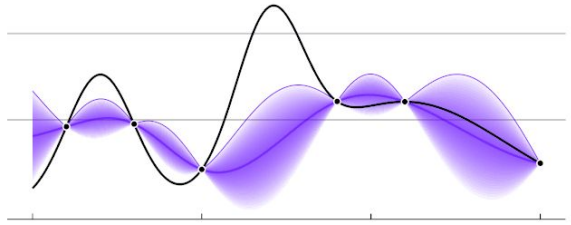
Next Time

Lecture: Using predictive UQ models for active learning and sequential decision making, including Bayesian optimization and optimal experimental design.

- Decision making under uncertainty.
- Active learning.
- Bayesian optimization.
- Optimal experimental design.
- Extensions: Bayesian algorithm execution.



"Generalizing Bayesian Optimization with Decision-theoretic Entropies", 2022

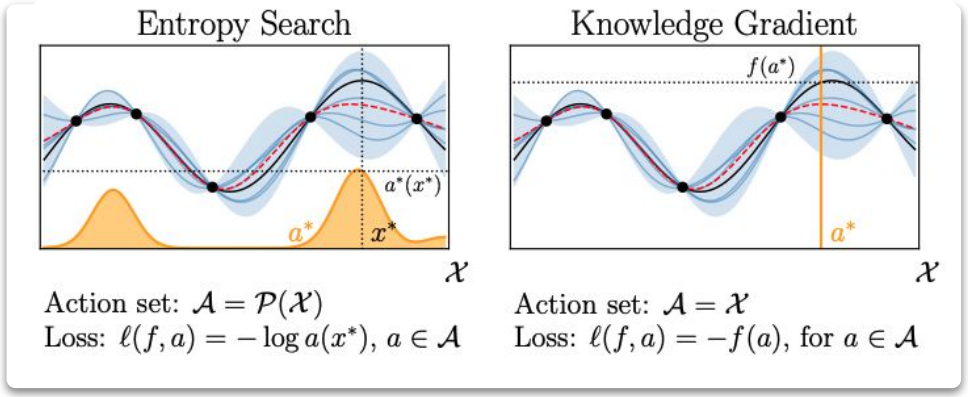


Bayesian Optimization, Wikipedia

Next Time

Lecture: Using predictive UQ models for active learning and sequential decision making, including Bayesian optimization and optimal experimental design.

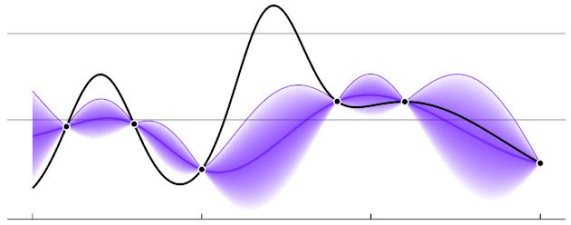
- Decision making under uncertainty.
- Active learning.
- Bayesian optimization.
- Optimal experimental design.
- Extensions: Bayesian algorithm execution.



"Generalizing Bayesian Optimization with Decision-theoretic Entropies", 2022

After:

- Paper presentations from students.



Bayesian Optimization, Wikipedia