LEVERAGING RELATED INSTANCES FOR BETTER PREDICTION

Siyuan Shan

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the the Department of Computer Science.

Chapel Hill
2023

ABSTRACT

Siyuan Shan: Leveraging Related Instances for Better Prediction
(Under the direction of Junier Bárbaro Oliva)

One fundamental task of machine learning is to predict output responses y from input data x. However, despite significant advances in the past decade, most current predictive models still only consider every single x in isolation when making predictions, which inevitably impacts model performance as the model may lose the opportunity to extract helpful information from other related instances to better predict x. This dissertation pushes the boundaries of machine learning research by explicitly taking advantage of related instances for better prediction. We find that leveraging multiple learned or intrinsically-related instances when making predictions in a data-driven and flexible manner is important for achieving good performance over a myriad of tasks.

When x is a single instance, we can flexibly find related instances based on similarity measurements. We develop algorithms that consider related neighborhood instances for a specific given x during prediction. Our assumption is that similar instances can be found near one another in an embedding space and they locally share a similar predictive function. We develop a model, Meta-Neighborhood, to learn a dictionary of neighbor points during training so that we can retrieve related instances from this dictionary during inference for improved classification and regression. Furthermore, this work is extended to Differentiable Wavetable Synthesis (DWTS) which leverages a dictionary of related basis waveforms for audio synthesis. We show that realistic audio can be synthesized by directly combining those basis waveforms.

Next, we consider the case where x is a given collection that contains multiple instances. In this case, x already included multiple related instances and we develop methods that learn how to exploit these related instances together to improve the prediction. Algorithms are developed to discover those instances more related to the prediction tasks and encourage the model to focus on these related instances for prediction. We first develop a transparent and human-understandable algorithm CKME that summarizes millions of instances into hundreds whilst being comparably accurate for single-cell set classification. Then an algorithm NRTSI for time series imputation is developed that treats the time series as a set and imputes missing data by leveraging those observed related data.

To my parents SSH and QXT. They instilled an interest in science and engineering and a desire to learn from an early age. To my beautiful and loving girlfriend, Madison, for her advice, her wisdom, and her support.

ACKNOWLEDGEMENTS

Each of the following people deserves a lot of credit for helping me complete this work. I appreciate each of them considerably more than the following brief sentences can express.

- Junier Bárbaro Oliva marked every page of this dissertation. He guided the thinking that went into the research and improved the writing of papers. This work is drastically improved thanks to his years of help and guidance.

- Natalie Stanley provided me the opportunity to work on the direction of single-cell data analysis and provided me with lots of instructions.

- Yang Li proposed several interesting research directions and ideas to me and provided help with writing.

- Christopher M. Bender proposed a helpful idea to improve my paper Meta-Neighborhoods.

- Hanoi Hantrakul and Jitong Chen provided me the opportunity to work on the direction of AI for audio and music.

- Haidong Yi provided help on running his CytoSet model as a baseline in my paper CKME.

- Vishal Athreya Baskaran provided his source codes to implement Random Fourier Features and Kernel Herding.

- Jolene Ranek helped me analyze the gradient map of single-cell features.

- Lamtharn Hantrakul, Jitong Chen, Matt Avent, and David Trevelyan helped to get a better understanding of digital signal processing and Wavetable Synthesis.

This work was also supported by various funding agencies:

- Teaching Assistant (TA) in the Department of Computer Science, UNC Chapel Hill
- ByteDance’s intern program
- Tiktok’s intern program
- NIH grant 1R01AA02687901A1
- NSF grant IIS2133595
- GPU provided by UNC ITS Research Computing

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

kNN	K-nearest Neighbors Algorithm
DWTS	Differentiable Wavetable Synthesis
MAML	Model-Agnostic Meta-Learning
DDSP	Differentiable Digital Signal Processing
RKHS	Reproducing Kernel Hilbert Space
MHA	Multi-head Attention

CHAPTER 1: OVERVIEW

1.1 Leveraging Related Neighbor Instances

Machine learning models are built to automatically exploit patterns and structures in training data to make predictions or decisions. The main objective of machine learning models is to infer the relationship between the input data $\mathbf{x}$ and its corresponding label y. Depending on applications, y could be a categorical variable for classification tasks such as image recognition, a real-valued scalar for regression tasks such as stock price prediction, or an action for reinforcement learning tasks. However, learning a function that directly infers y only from $\mathbf{x}$ may be challenging. As a motivating analogy, consider the following: when faced with a new unseen instance, $\mathbf{x}$, humans not only analyze $\mathbf{x}$, but also leverage previous examples in their memory that are similar to $\mathbf{x}$ to make a prediction [97]. The first "predicting by analysis" method is the philosophy behind the popular deep learning methods where the models analyze the input by feature extraction. The second "predicting by comparison" method constitutes the success of the classic K-nearest neighbors algorithm (kNN) algorithm that makes decisions by first finding related neighbors and taking a majority vote on the outcomes of these neighbors. However, the latter method is largely neglected in a world dominated by deep learning approaches nowadays.

In this dissertation, we develop novel algorithms that enable deep learning models to also consider related neighbor instances to achieve better prediction. Specifically, two methods are developed to consider related neighbors for predicting y. In the first method, given a deep learning model $f_\phi(\cdot)$ parameterized by ϕ, we allow ϕ to be adjusted using neighbors related to a specific input x during inference and use the model with adjusted parameter ϕ^* to predict on x, i.e. $y = f_{\phi^*}(x)$. In this way, our prediction can implicitly depend on the neighbor instances. In the second method, the outputs are expressed as a weighted average of related neighbors and the averaging weights are predicted by a neural network f. This paradigm allows related neighbor points to have a direct impact on the final predictions. For both methods, we treat these related neighbor instances as a part of all the learnable parameters in our model and optimize them end-to-end with the neural networks using gradient descent. In this way, we guarantee that the learned neighbor instances are compatible with our neural networks. Note that our methods still maintain the feature learning capacity of deep learning approaches as we leverage related neighbor points only to enhance the predictions.

By leveraging these related neighbor points, higher-quality predictions can be achieved as the model can gather information from similar points to gain additional context. As a by-product, we find the learned related neighbors provide insights into understanding how the decision is made. This is because the related instances explain the behavior of machine learning models or the underlying data distribution. For instance, we find that different related instances may represent different concepts of the datasets, i.e. some represent speedboats while others represent cargo ships.

In the next section, we will discuss how this idea to leverage related instances can be extended to cases where the input data is a collection of features rather than a single feature.

1.2 Leveraging Related Points in a Collection

Unlike traditional machine learning tasks where the input data can be succinctly expressed as a single multi-dimensional feature vector, there are other more complicated applications, such as population statistics estimation [223], point cloud classification [114] and object detection [206], image registration [173], time series classification [74, 199], multiarray signal processing [120, 110, 145], and multi-frame video classification [53] where the input is a collection of feature vectors. Learning over this type of input data is challenging as the model not only needs to characterize and compare across multiple collections but also must reason and identify the interactions between the elements in each collection. Also, handling a collection of multi-dimensional feature vectors incurs higher computational overhead and further obscures the decision-making process. For these applications, when the instances in the collection are unordered, this collection is a set of instances. If the instances are ordered, this collection can be seen as a sequence of instances.

In the same vein as the discussion in the previous section, we also develop algorithms to leverage the related instances to achieve better predictions in the case where the input data is a collection. Specifically, we propose methods to find a subset of related instances belonging to the original input collection for better predictions. Note that a slight difference from the previous section is that we do not find other instances related to the input. Instead, in this section, we aim to find related instances inside the input collection itself. Our methods are based on the assumption that not all instances in a collection are equally important to predict the outcome of the collection. For instance, consider the case where the task is to predict whether a point cloud (i.e. a set of points in a three-dimensional space) represents an airplane, the points in the area of wings and engines contribute more semantic clues for our specific prediction task than those in other areas.

Concretely, we propose an explicit method as well as an implicit method to find related instances. Firstly, we propose a method that builds on kernel methods and herding algorithms to explicitly find only hundreds of related points from a collection of millions of points while maintaining a high clinical outcome classification accuracy for single-cell data. Our proposed approach significantly reduces the computational cost of considering millions of points simultaneously for classification and enables better interpretability as the decision is made only on hundreds of points. Secondly, we propose an implicit method to discover related instances for time series imputation. To impute missing data, the model needs to collect information from the observed time points. However, not all the observed time points are equally important for predicting a specific missing value. To find those related instances that are more important, we adopt an attention mechanism to allow the model to automatically infer the relationship between different time points, and thus related instances are learned implicitly.

Below, I will introduce the necessary background knowledge and our proposed methods in detail. Then the applications of the proposed approaches will be discussed.

1.3 Background

In this chapter, I introduce the necessary background for our proposed methods. More specifically, my work Meta-Neighborhoods [170] requires an understanding of a well-known meta-learning technique MAML [56]. Then we extend the idea of Meta-Neighborhoods for audio synthesis and propose a method called Differentiable Wavetable Synthesis (DWTS) [169]. DWTS is built on a pioneering work DDSP [44] that combines the interpretable structure of classical DSP elements (such as filters, oscillators, reverberation, etc.) with the expressivity of deep learning. Also, both Meta-Neighborhoods and Differentiable Wavetable Synthesis adopt the attention mechanisms. NRTSI [171] employ a Transformer architecture [191] to implicitly identify instances that are related to the imputation targets. CKME [168] and its application for sketching single-cell samples [9] use *kernel mean embedding* [140] to sub-select sample-sets. Since both NRTSI and CKME are learning over sets, we also introduce here several properties of sets and existing methods that can handle this type of data.

1.3.1 Model-Agnostic Meta-Learning (MAML)

Our work Meta-Neighborhood in Chapter 2 is built on MAML [56], a model-agnostic meta-learning method. The main goal of meta-learning is to learn quickly on new tasks with the help of knowledge on the old tasks. MAML [56] aims to learn a set of good initial model parameters that can be quickly fine-tuned to achieve good performance for a new task. Therefore, knowledge about the old tasks is effectively summarized by this set of initial parameters. MAML learns the initial parameters ϕ by optimizing the following loss $L(\phi)$ with gradient descent:

$$L(\phi) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(\phi_i) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(\phi - \alpha \nabla_\phi \mathcal{L}_{\mathcal{T}_i}(\phi)), \tag{1.1}$$

where $\mathcal{T}_i$ is a task sampled from a distribution of tasks $p(\mathcal{T})$, $\mathcal{L}_{\mathcal{T}_i}(\phi_i)$ means the loss on the task $\mathcal{T}_i$ evaluated with the model parameter ϕ_i. From (1.1), we can see that MAML first fine-tunes ϕ for every task as $\phi_i = \phi - \alpha\nabla_\phi\mathcal{L}_{\mathcal{T}_i}(\phi)$, which is called the inner-loop optimization. Then the model computes the average loss across all tasks, minimizing which can learn a good initial parameter ϕ that can be easily fine-tuned for all tasks. This step is often called outer-loop optimization.

1.3.2 Differentiable Digital Signal Processing (DDSP)

In Chapter 3, we extend the idea of Meta-Neighborhood and propose a method Differentiable Wavetable Synthesis, which is built on a previous work DDSP [44]. DDSP described a model that introduces modules with strong inductive biases from the traditional digital signal processing library (e.g. filters, oscillator, reverberation, etc.) to deep learning. As all the modules are differentiable, they can be easily combined with neural networks for end-to-end learning. The core module of DDSP is an additive synthesizer.

In additive synthesis [136], you can build a sound from scratch by combining multiple sinusoidal waves of different amplitudes and frequencies. This way of sound synthesis is intuitive because the sounds that are heard in everyday life do not contain a single frequency. Instead, they consist of a sum of pure single frequencies as illustrated by Fig. 1.1. The timbre of the synthesized sound is determined by the amplitudes of every single sinusoidal wave (single frequency component). For example, Fig. 1.2 illustrates the spectrograms of four instruments. We can see that each different instrument has a different summation of pure single frequencies, resulting in different timbres. For instance, trumpets sound different from flutes because the high-frequency components of trumpets have higher amplitudes.

More formally, similar to the summation mechanism shown in Fig. 1.1, the additive synthesizer in DDSP combines K sinusoidal functions with different predefined frequencies to output the synthesized signal $s(n)$ over discrete time steps, n:

$$s(n) = \sum_{k=1}^{K} A_k(n)\sin(\phi_k(n)), \tag{1.2}$$

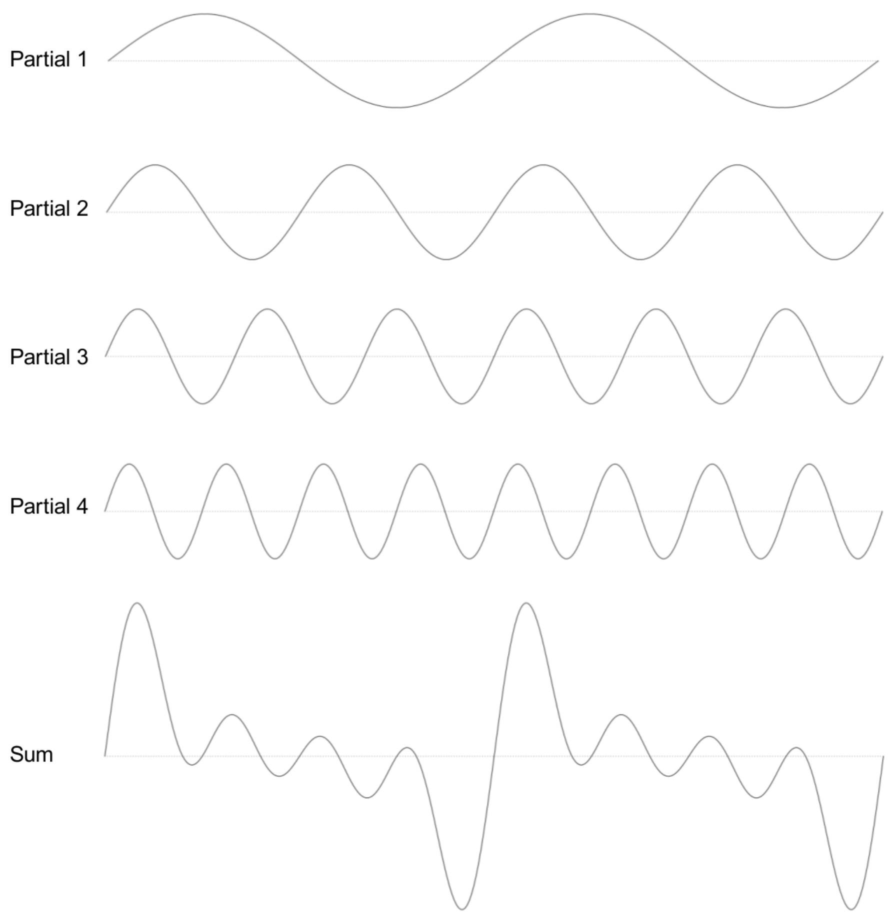


Figure 1.1: Illustration of additive synthesis. Image is taken from [1].

where $A_k(n)$ is predicted by a neural network and represents the time-varying amplitude of the k-th sinusoidal components, $\phi_k(n) = 2\pi \sum_{m=0}^{n} f_k(m)$ is the corresponding instantaneous phase obtained by integrating the predefined instantaneous frequency f_k. DDSP assumes $f_k = kf_0(n)$ where $f_0(n)$ is the time-varying fundamental frequency. Therefore, the synthesized signal only contains harmonics with integer multiples of the fundamental frequency.

1.3.3 Attention Mechanisms

In neural networks, attention is a technique that mimics cognitive attention. The effect emphasizes some parts of the input data while ignoring other parts. The motivation is that the network should devote more focus to the small, but important, parts of the data. Learning which part of

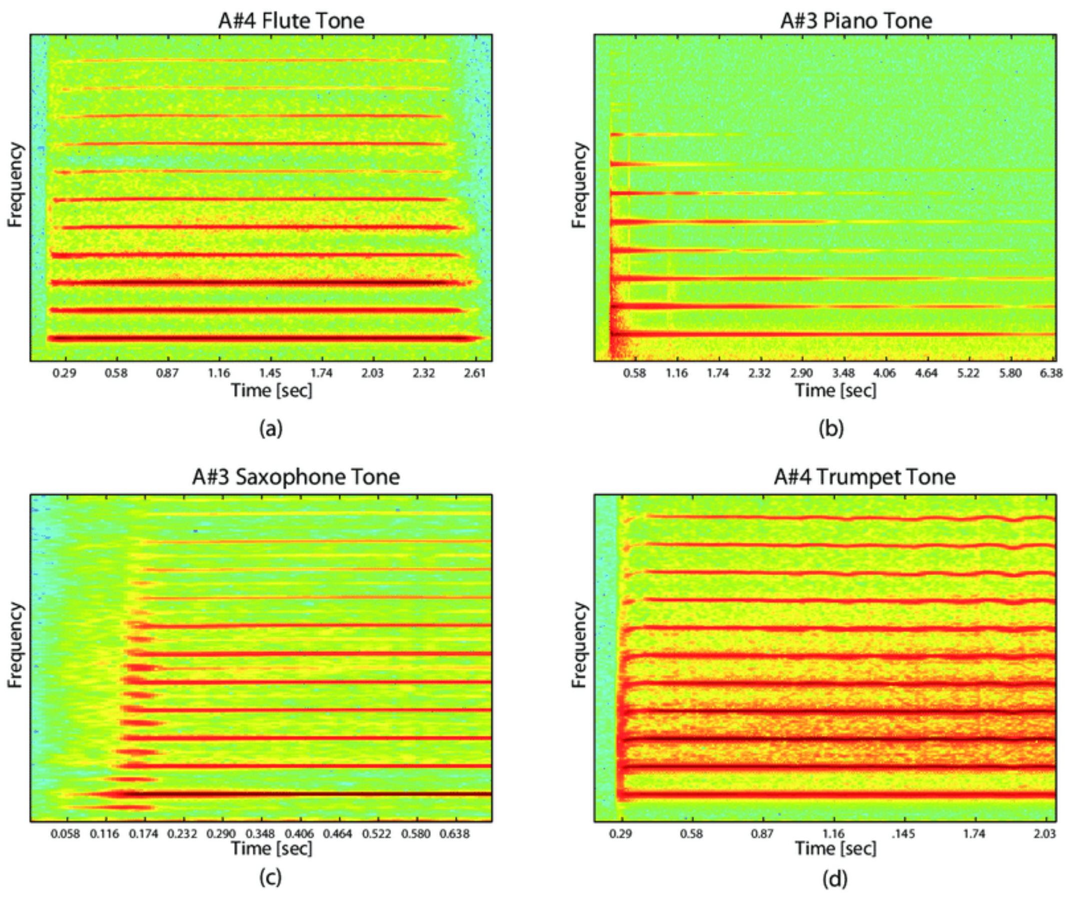


Figure 1.2: Illustration of different compositions of harmonic components of different instruments. Image is taken from [21].

the data is more important than another depends on the context, and this is trained by gradient descent.

Attention mechanisms are mainly used to retrieve relevant information from a set of candidates $\{k_1, k_2, ..., k_n\}$ where k_j is a high-dimensional feature vector. Given a query vector x_i, we first compute the similarity of x_i to all the candidates using a dot product as $x_i^T \cdot k_j$. Then, the softmax function is applied to compute the attention weights $\omega_{i,j}$ of all candidates as

$$\omega_{i,j} = \frac{e^{x_i^T \cdot k_j} / T}{\sum_{m=1}^{n} \gamma e^{x_i^T \cdot k_m} / T}, \tag{1.3}$$

where T is usually known as the temperature parameter of the softmax function that controls the sparseness of the attention pattern. A smaller T leads to a sparser attention weight. By default, T

is equal to 1. Finally, the query result is computed as a weighted average of all the candidates, i.e.

$$q_i = \sum_{j=1}^{n} \omega_{i,j} k_j. \tag{1.4}$$

Note that it is guaranteed that the attention weights sum up to 1 according to the definition of the softmax function. Alternatively, in some applications, we may be interested in retrieving the values corresponding to these candidates rather than the candidates themselves. In such a case, the query result is

$$q_i = \sum_{j=1}^{n} \omega_{i,j} v_j, \tag{1.5}$$

where v_j is the value of the candidate k_j.

1.3.4 Transformer

Before the introduction of Transformer models [191], recurrent neural networks [72, 34] and convolution neural networks [100, 96] dominated sequence learning tasks. Though there are some earlier efforts [8] that combine attention mechanisms with recurrent models to improve the performance, Transformer models show that an attention mechanism is all you need to achieve state-of-the-art performance on the sequence learning tasks. This is because the attention mechanism itself allows direct access to any position in the input sequence and effectively solves the information loss problem of recurrent models. Transformer models [191] have revolutionized both natural language processing fields [41] and computer vision fields [42] due to their strong capacity to model interactions between L d-dimensional input tokens $\mathbf{X} \in \mathbb{R}^{L \times d}$ with attention,

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \in \mathbb{R}^{L \times d_v}, \tag{1.6}$$

$$Q = \mathbf{X}\mathbf{W}_q \in \mathbb{R}^{L \times d_k}, \tag{1.7}$$

$$K = \mathbf{X}\mathbf{W}_k \in \mathbb{R}^{L \times d_k}, \tag{1.8}$$

$$V = \mathbf{X}\mathbf{W}_v \in \mathbb{R}^{L\times d_v}, \tag{1.9}$$

where $\mathbf{W}_q \in \mathbb{R}^{d\times d_k}, \mathbf{W}_k \in \mathbb{R}^{d\times d_k}, \mathbf{W}_v \in \mathbb{R}^{d\times d_v}$ are learnable parameters to compute the query vector Q, the key vector K, and the value vector V. The hyperparameters d_k and d_v are the dimensions of the key vector and the value vector. This attention mechanism is further improved by using multi-head attention (MHA), which uses multiple independent heads to capture different features. MHA is a module for attention mechanisms that run through an attention mechanism several times in parallel. The independent attention outputs are then concatenated and linearly transformed into the expected dimension, which can be written as

$$\text{MultiHead}(Q, K, V) = O\mathbf{W}_0 \in \mathbb{R}^{L\times d}, \tag{1.10}$$

$$O = \text{Concat}(\text{head}_1, ..., \text{head}_h) \in \mathbb{R}^{L\times hd_v} \tag{1.11}$$

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) \in \mathbb{R}^{L\times d_v}, \tag{1.12}$$

$$Q_i = \mathbf{X}\mathbf{W}_q^i \in \mathbb{R}^{L\times d_k}, \tag{1.13}$$

$$K_i = \mathbf{X}\mathbf{W}_k^i \in \mathbb{R}^{L\times d_k}, \tag{1.14}$$

$$V_i = \mathbf{X}\mathbf{W}_v^i \in \mathbb{R}^{L\times d_v}, \tag{1.15}$$

where $\mathbf{W}_q^i \in \mathbb{R}^{d\times d_k}$, $\mathbf{W}_k^i \in \mathbb{R}^{d\times d_k}$, $\mathbf{W}_v^i \in \mathbb{R}^{d\times d_v}$, and $\mathbf{W}_O \in \mathbb{R}^{hd_v\times d}$ are learnable projection matrices. Attention is usually implemented as scaled dot-product attention as in (1.6). Intuitively, multiple attention heads allow for attending to parts of the sequence differently (e.g. longer-term dependencies versus shorter-term dependencies [228]). Unlike recurrent neural networks that are naturally aware of the sequence order of the input tokens, Transformer models do not have such a concept as all the input tokens are processed in parallel. To inject the order information,

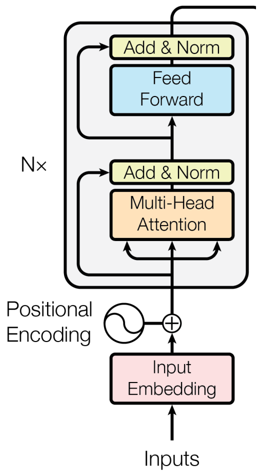


Figure 1.3: Transformer Encoder Architecture.

Transformer models introduce positional encodings,

$$\mathrm{PE}_{(\mathrm{pos},2i)} = \sin(\mathrm{pos}/10000^{2i/d})$$
$$\mathrm{PE}_{(\mathrm{pos},2i+1)} = \cos(\mathrm{pos}/10000^{2i/d}),$$

where pos is the position and i is the dimension.

The full Transformer encoder architecture is illustrated in Fig 1.3. It stacks N blocks to enhance feature extraction capacity and each block consists of a MHA layer, normalization layers, and a feed-forward layer. Skip connection is used to facilitate the gradient flow.

1.3.5 Kernel Mean Embedding

Please see Muandet et al. [140] for a thorough review of kernel mean embeddings. Kernel methods have achieved great success in many distinct machine learning tasks, including classification [35], regression [196], and dimensionality reduction [137]. They utilize a positive definitive kernel function $\mathrm{k} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ [1], which induces a reproducing kernel Hilbert space (RKHS) (e.g. see Berlinet & Thomas-Agnan [12] for further details). Kernels have also been deployed for

[1]Note that kernels may be defined over non-real domains, this is omitted for simplicity.

representing a distribution, p, with the *kernel mean embedding* $\mu_p : \mathbb{R}^d \mapsto \mathbb{R}$:

$$\mu_p(\cdot) \equiv \mathbb{E}_{x \sim p}[\mathtt{k}(x, \cdot)]. \tag{1.16}$$

Note that μ_p is itself a function. For "*characteristic*" kernels, $\mathtt{k}$, such as the common radial-basis function (RBF) kernel $\mathtt{k}(x, x') = \exp(-\frac{1}{2\gamma}||x - x'||^2)$, the kernel mean embedding will be unique to its distribution; i.e., for characteristic kernels, $||\mu_p - \mu_q|| = 0$ if and only if $p = q$. In general, the distance[2] $||\mu_p - \mu_q||$ induces a divergence, the maximum mean discrepancy (MMD) [67], between distributions.

1.3.6 Permutation Invariant Mappings on Sets

A set is a collection that does not impose order among its elements. Models over sets *must* preserve this property. We list the commonly used terminology for set modeling below. We denote a set as $\mathbf{x} = \{x_i\}_{i=1}^N \in \mathcal{X}^N$, where N is the cardinality of the set and the calligraphic letter $\mathcal{X}$ represents the domain of each element x_i.

Definition 1. *(Permutation Equivariant) Let $f : \mathcal{X}^N \rightarrow \mathcal{Y}^N$ be a function, then f is permutation equivariant iff for any permutation $\pi(\cdot)$, $f(\pi(\mathbf{x})) = \pi(f(\mathbf{x}))$.*

Definition 2. *(Permutation Invariant) Let $f : \mathcal{X}^N \rightarrow \mathcal{Y}$ be a function, then f is permutation invariant iff for any permutation $\pi(\cdot)$, $f(\pi(\mathbf{x})) = f(\mathbf{x})$.*

A naive method to encourage permutation invariance is to augment the training data with randomly permuted sets and treat them as sequences. One could then train a neural network mapping the permuted inputs to the same output. Due to the universal approximation ability of neural networks, the final model could be invariant to permutations given an infinite amount of training data and model capacity. However, this simple approach does not guarantee invariance for real-world finite datasets. As pointed out by Vinyals et al. [193], the order cannot be discarded for a sequence model.

[2]In the RKHS norm.

DeepSet [223] proves that any permutation invariant function for a set with a finite number of elements can be decomposed as $\rho(\sum_{x \in \mathcal{X}^N} \phi(x))$, where the summation is over the set elements. Based on this decomposition, they propose using two neural networks for both ρ and ϕ to learn flexible permutation invariant functions. They also propose an equivariant model, where independent processing combined with a pooling operation is used to capture the intradependencies. Although deep sets are universal approximators, there are some constraints with respect to the dimensionality of latent representation as shown by Wagstaff et al. [197].

Set Transformer [102] proposes to use an attention mechanism over set elements to model the intradependencies between each pair of elements. Since the attention is a weighted sum over all set elements, this operation is naturally equivariant. They also propose an attention-based pooling operation to achieve invariant representations.

1.4 Thesis Structure

This dissertation is structured as follows:

1. In Chapters 2 and 3, we focus on the case where the input contains only a single instance and demonstrate the technical development for better prediction.

    - Chapter 2: We develop a meta-learning pipeline to learn a dictionary of related instances to achieve better image classification and tabular data regression performance. During inference, related instances are retrieved via an attention mechanism to adjust model parameters to improve performance. A similar idea of learning a dictionary of related instances for audio synthesis is proposed in Chapter 3.

    - Chapter 3: We propose an audio synthesis method Differentiable Wavetable Synthesis (DWTS) that synthesizes by reusing a set of learned instance waveforms. Our work can synthesize harmonic audio in a controllable and interpretable manner. DWTS has a fast synthesis speed and superior one-shot learning performance.

2. In Chapters 4 and 5, we consider the case where the input data is a collection of instances and develop methods to jointly consider the given set of related points to make a better prediction on the collection.

    - Chapter 4: We develop an interpretable machine learning model CKME for classifying single-cell sample sets, which is a collection of feature vectors of a set of cells. CKME uses kernel mean embedding to build permutation-invariant featurization of each individual sample set.
    - Chapter 5: We develop a method NRTSI for time series imputation. Unlike traditional approaches that view time series as a sequence, we view a time series as a set. This allows us to develop a hierarchical imputation procedure and use attention mechanisms to better impute missing data.

CHAPTER 2: LEVERAGING RELATED NEIGHBORS FOR IMPROVED PREDICTION

2.1 Notation

- x: an input data $x \in \mathbb{R}^{d_i}$.
- y: an output response $y \in \mathbb{R}^{d_o}$.
- $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$: a training set with N input-output pairs.
- $\omega(x_i, k_j)$: attention weight between query x_i and key k_j.
- $M = \{(k_j, v_j)\}_{j=1}^{S}$: a dictionary that stores neighbors where S is the number of induced neighbors. Just like the real training set $\mathcal{D}$, the dictionary stores input-output pairs as key-value pairs (k_j,v_j)
- f_ϕ: a neural network with parameter ϕ.

2.2 Introduction

Neural networks assume that a set of *fixed* parameters ϕ determines the predictive distribution, i.e., $p(y \mid x; \phi)$. The training process estimates ϕ and then discards the training data completely, as the learned parameters ϕ are responsible for the ultimate prediction. This paradigm has proven effective, however, it leaves the entire burden on learning a complex predictive distribution over potentially large support. The well-known k-nearest neighbor (kNN) estimator, in contrast, often achieves surprisingly good results by leveraging neighbors from the training data, which reduces the problem to a much simpler local manifold.

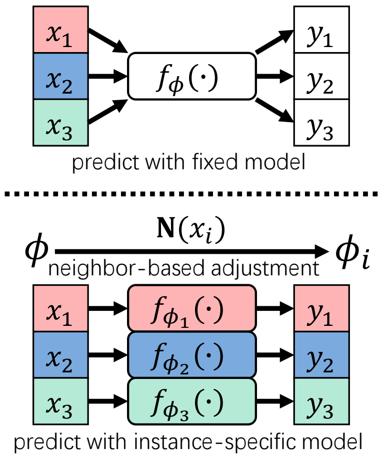


Figure 2.1: Top: traditional parametric models. Bottom: our per-instance-adapted model. $\mathbf{N}(x_i)$ denotes the set of neighbors of instance x_i.

In this work, we improve traditional neural networks with related neighbors and propose a method called Meta-Neighborhoods. The main body of Meta-Neighborhoods is a parametric neural network, but we adapt its parameters to a local neighborhood in a non-parametric scheme. The prediction is made on the local manifold by the adapted model. Fig. 2.1 illustrates the difference between traditional parametric models and the proposed model. Inspired by the success of inducing point methods from sparse Gaussian process literature [178, 188] to alleviate the storage burden and reduce time complexity, we learn induced neighborhoods, which summarize the training data into a much smaller dictionary. The induced neighborhoods and the neural network parameters are learned jointly.

Our model is also closely related to locally linear embeddings [161], which reconstructs the non-linear manifold with locally linear approximation around each neighborhood. In our method, we adapt an initial model (not necessarily linear) to local neighborhoods.

Our method imposes challenges of adapting the initial model since the local neighborhoods usually do not contain enough training instances to independently adapt the model, and the induced neighborhoods contain even fewer instances. Inspired by the few-shot and meta-learning literature [56], we propose a meta-learning-based training mechanism, where we learn an initial model so that it adapts to the local neighborhood after only several fine-tuning steps over a few instances inside the neighborhood.

The prediction process of our model proceeds as follows. An input x is first paired with its neighbors by querying the induced dictionary. The initial model is adapted to its neighborhood by fine-tuning several steps on the neighbors. We then predict the target y using the adapted model.

Our contributions are as follows:

- We improve neural networks by considering related neighbor points.
- We propose Meta-Neighborhoods to jointly learn the induced neighborhoods and an adaptive initial model, which can adapt its parameters to a local neighborhood according to the input through both fine-tuning and a proposed instance-wise modulation scheme, iFiLM.
- Extensive empirical studies demonstrate the superiority of Meta-Neighborhoods for both regression and classification tasks.
- We empirically find the induced neighbors are semantically meaningful: they represent informative boundary cases on both realistic and toy datasets; they also capture sub-category concepts even though such information is not given during training.

2.3 Problem Formulation

Given a training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ with N input-target pairs, we learn a discriminative model $f_\phi(x)$ and a dictionary $M = \{(k_j, v_j)\}_{j=1}^S$ jointly from $\mathcal{D}$. The learned dictionary stores the neighbors induced from the training set, where S is the number of induced neighbors. Just like the real training set $\mathcal{D}$, the dictionary stores input-target pairs as key-value pairs (k_j,v_j), where both the keys and the values are learned end-to-end with the model parameters ϕ in a meta-learning framework. For classification tasks, v_j is a vector representing the class probabilities while for regression tasks v_j is the regression target. In the following text, we will use the terms "induced neighbors" and "learnable neighbors" interchangeably.

2.4 Background

Our model is built on the well-known meta-learning algorithm Model-Agnostic Meta-Learning (MAML) [56]. Meta-learning, also known as learning to learn, allows machine learning models to learn new concepts and skills quickly and efficiently with a limited number of training examples [78]. This is in contrast to popular machine learning models nowadays such as neural networks, which are data-hungry and requires many iterations to converge. Also, unlike contemporary machine learning models that only learn a single specific task, meta-learning algorithms learn several tasks simultaneously and can adapt to generalize to new tasks that are not seen during training. The tasks can be any well-defined family of machine learning problems such as supervised learning and reinforcement learning. There are three categories of meta-learning algorithms, namely metric-based, model-based and optimization-based. Metric-based methods predict by computing the similarity between a query example and a set of support examples in the new task and aggregate the output responses according to the similarity. This is similar to nearest neighbor algorithms like kNN. With several different ways to compute the similarity, representative metric-based works are Matching Networks [194] and Prototypical Networks [177]. Model-based algorithms either use external memory storage to facilitate the learning of neural networks [166] or perform rapid parameter updates by its internal architecture [141].

MAML belongs to the third categorical which is optimization-based. MAML [56] aims to learn a set of good initial model parameters that can be quickly fine-tuned to achieve good performance for a new task. During inference, MAML fine-tunes the initial parameters ϕ by performing gradient descents to minimize the task loss $\mathcal{L}_{\mathcal{T}_i}$:

$$\phi_i \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_{\mathcal{T}_i}(\phi), \tag{2.1}$$

where $\mathcal{T}_i$ is a task sampled from a distribution of tasks $p(\mathcal{T})$. MAML uses few-shot learning to test the meta-learning performance where each task has a support set with several labeled data instances and a query set with several unlabeled data instances whose labels need to be predicted.

In our model, we follow a similar fashion but the support set in our model is a set of instances related to the input instance and the query set only contains that single input instance.

2.5 Methods

2.5.1 Prediction with Induced Neighborhoods

In this section, we assume access to the learned neighborhoods in M and the learned model f_ϕ. Different from the conventional parametric setting, where the learned model is employed directly to predict the target, we adapt the model according to the neighborhoods retrieved from M and the adapted model is the one responsible for making predictions. Specifically, for a test data instance x_i, relevant entries in M are retrieved in a soft-attention manner by comparing x_i to k_j via an attention function ω. The retrieved entries are then utilized to fine-tune f_ϕ following

$$\phi_i \leftarrow \phi - \alpha \nabla_\phi \sum_{j=1}^{S} \omega(x_i, k_j) L(f_\phi(k_j), v_j), \tag{2.2}$$

where α is the fine-tuning step size. We set α to be a learnable variable in our work. Note we weight the loss terms of all dictionary entries by their similarities to the input test data x_i. The intuition is that nearby neighbors play a more important role in placing x_i into the correct local manifold. Since the model f_ϕ is specially trained in a meta-learning scheme, it adapts to the local neighborhoods after only a few fine-tuning steps.

To better understand our method, we draw connections to other well-known techniques. The above prediction process is similar to a one-step EM algorithm. Specifically, the dictionary querying step is an analogy to the Expectation step, which determines the latent variables (in our case, the neighborhood assignment). The fine-tuning step is similar to the Maximization step, which maximizes the expected log-likelihood. We can also view this process from a Bayesian perspective, where the initial parameter ϕ is an empirical prior estimated from data; posteriors are de-

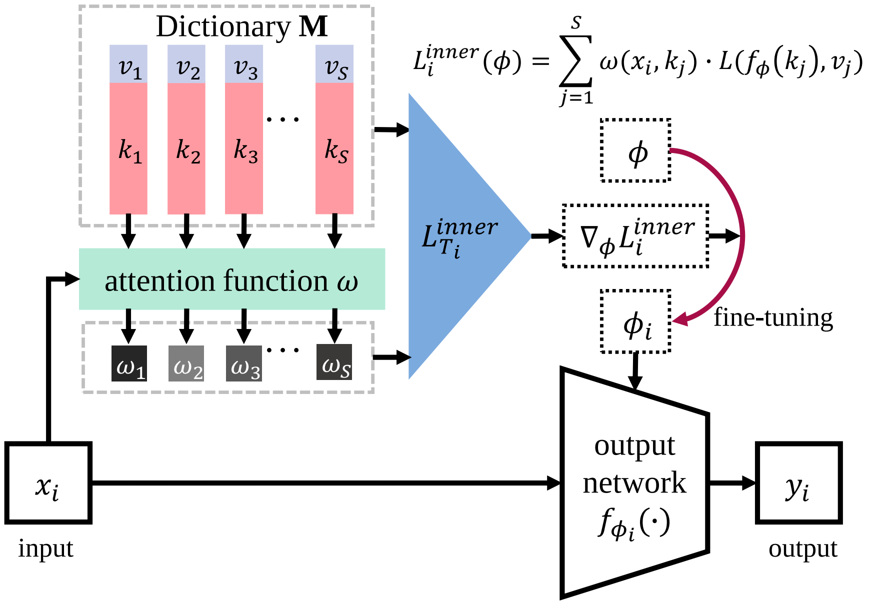

Dictionary **M**
v_1 v_2 v_3 v_S
k_1 k_2 k_3 k_S
attention function ω
ω_1 ω_2 ω_3 ω_S
$L_i^{inner}(\phi) = \sum_{j=1}^{S} \omega(x_i, k_j) \cdot L(f_\phi(k_j), v_j)$
$L_{T_i}^{inner}$
ϕ
$\nabla_\phi L_i^{inner}$
ϕ_i
fine-tuning
x_i
input
output network $f_{\phi_i}(\cdot)$
y_i
output

Figure 2.2: Model overview.

rived from neighbors following the fine-tuning steps. The predictive distribution with posterior is used for the final predictions.

Algorithm 1 META-NEIGHBORHOODS: TRAINING PHASE

```
Require: $\omega$: similarity metric, $\eta$: outer loop learning rate
 1: Initialize $\theta, \phi, \alpha, M = \{(k_j, v_j)\}_{j=1}^S$
 2: while not done do
 3:     Sample a batch of training data $\{(x_i, y_i)\}_{i=1}^B$
 4:     for all $(x_i, y_i)$ in current batch do
 5:         Compute the feature vector $z_i = \mu_\theta(x_i)$
 6:         Compute $\mathcal{L}_i^{\text{inner}}(\phi) = \sum_{j=1}^S \omega(z_i, k_j) L(f_\phi(k_j), v_j)$
 7:         Finetune $\phi$: $\phi_i = \phi - \alpha \nabla_\phi \mathcal{L}_i^{\text{inner}}(\phi)$
 8:     end for
 9:     Compute $\mathcal{L}^{\text{meta}}(\phi, \theta, M, \alpha) = \frac{1}{B} \sum_{i=1}^B L(f_{\phi_i}(\mu_\theta(x_i)), y_i)$
10:     Update model parameters $\Theta = \{\theta, \phi, M, \alpha\}$
        using gradient descent as $\Theta \leftarrow \Theta - \eta \nabla_\Theta \mathcal{L}^{\text{meta}}$
11: end while
```

2.5.2 Joint Meta Learning

Above, we assume access to a given dictionary M and a given model f_ϕ. In this section, we describe our meta-learning mechanism to train them jointly. The training strategy of Meta-

Neighborhoods resembles MAML [56] in that both adopt a nested optimization procedure, which involves an inner loop update and an outer loop update in each iteration. Note that in contrast to MAML, we are solving a general discriminative task rather than a few-shot task [208]. Given a batch of training data $\{x_i, y_i\}_{i=1}^B$ with a batch size B, in the inner loop we fine-tune the initial parameter ϕ to ϕ_i in a similar fashion to (2.2). With ϕ individually fine-tuned for each training data x_i using its corresponding neighborhoods, we then jointly train the model parameter ϕ, the dictionary M as well as the inner loop learning rate α in the outer loop using the following meta-objective function

$$\mathcal{L}^{\text{meta}}(\phi, M, \alpha) = \frac{1}{B}\sum_{i=1}^{B} L(f_{\phi_i}(x_i), y_i) = \frac{1}{B}\sum_{i=1}^{B} L(f_{\phi-\alpha\nabla_\phi \mathcal{L}_i^{\text{inner}}}(x_i), y_i), \tag{2.3}$$

where $\mathcal{L}_i^{\text{inner}}(\phi) = \sum_{j=1}^{S} \omega(x_i, k_j) L(f_\phi(k_j), v_j)$ according to (2.2). We set α to be a learnable scalar or diagonal matrix. $\mathcal{L}^{\text{meta}}$ encourages learning shared ϕ, M, and α that are widely applicable for data with the same distribution as the training data. An overview of our model is shown in Fig. 2.2.

Parameter ϕ serves as initial weights that can be quickly adapted to a specified neighborhood. This meta-training scheme effectively tackles the overfitting problem caused by the limited number of finetuning instances, as it explicitly optimizes the generalization performance after fine-tuning.

For high-dimensional inputs such as images, learning k_j in the input space could be prohibitive. Therefore, we employ a feature extractor μ_θ to extract the feature embedding $z_i = \mu_\theta(x_i)$ for each x_i and learn k_j in the embedding space. We accordingly modify (2.2) to

$$\phi_i \leftarrow \phi - \alpha\nabla_\phi \sum_{j=1}^{S} \omega(\mu_\theta(x_i), k_j) L(f_\phi(k_j), v_j), \tag{2.4}$$

where the attention function ω is employed in embedding space. The meta-objective is accordingly modified as $\mathcal{L}^{\text{meta}}(\phi, \theta, M, \alpha) = \frac{1}{B}\sum_{i=1}^{B} L(f_{\phi_i}(\mu_\theta(x_i)), y_i)$. We train θ and other learnable parameters jointly. Note that the model without a feature extractor can be viewed as a special case where μ_θ is an identity mapping. The pseudocode of our training algorithm is given in Algorithm 1.

It is also desirable to adjust μ_θ per instance. However, when μ_θ is a deep convolution neural network, tuning the entire feature extractor μ_θ is computationally expensive. Inspired by FiLM [149], we propose instance-wise FiLM (iFiLM) that adjusts the batch normalization layers individually for each instance. Suppose $a^l \in \mathbb{R}^{B\times C^l \times W^l \times H^l}$ is the output of the l_{th} Batch Normalization layer **BN**^{l}, where B is batch size, C^l is the number of channels, W^l and H^l are the feature map width and height. Along with each **BN**^{l}, we define a learnable dictionary $M^l = \{k_j^l, \gamma_j^l, \beta_j^l\}_{j=1}^{S_l}$ of size S^l. k_j^l are the keys used for querying. $\gamma_j^l, \beta_j^l \in \mathbb{R}^{C^l}$ represent the scale and shift parameters used for adaptation respectively. When querying M^l, the outputs a^l are first aggregated across their spatial dimensions using global pooling, i.e. $g^l = \textbf{GlobalAvgPool}(a^l) \in \mathbb{R}^{B\times C^l}$. Then, the instance-wise adaptation parameters $\widehat{\gamma_i^l}$ and $\widehat{\beta_i^l}$ are computed as

$$\widehat{\gamma_i^l} = \sum_{j=1}^{S^l} \omega(g_i^l, k_j^l)\gamma_j^l \in \mathbb{R}^{C^l} \qquad \widehat{\beta_i^l} = \sum_{j=1}^{S^l} \omega(g_i^l, k_j^l)\beta_j^l \in \mathbb{R}^{C^l}, \tag{2.5}$$

where ω is defined as in (2.2) and $i \in \{1, 2, \ldots, B\}$. Following FiLM [149], each individual activation a_i^l is then adapted with an affine transformation $\widehat{\gamma_i^l} \cdot a_i^l + \widehat{\beta_i^l}$. Note the transformation is agnostic to spatial position, which helps to reduce the number of learnable parameters in the dictionary M.

2.5.3 Other Details and Considerations

Above, we have discussed how we learn related instances and leverage them to obtain a more flexible model. In this section, we discuss further implementation details that enable successful model training. We also motivate our method from the perspective of k-nearest neighbor (kNN).

Similarity Metrics To implement the attention function ω in (2.2)(2.4)(2.5), we need a similarity metric to compare an input x_i with each key k_j. We consider two types of metrics, cosine similarity and negative Euclidean distance. We find cosine similarity works better than negative Euclidean distance in the case of high dimensions while the opposite is true in the case of low dimensions. The similarities of x_i to all keys are normalized using a softmax function with a temperature parameter T [71], i.e.,

$$\omega(x_i, k_j) = \frac{\exp(\text{sim}(x_i, k_j)/T)}{\sum_{s=1}^{S} \exp(\text{sim}(x_i, k_s)/T)}, \tag{2.6}$$

where $\text{sim}(\cdot)$ represents the similarity metric.

Initialization of the Dictionary Since we use a similarity-based attention function ω in (2.4), we would like to initialize the key k_j to have a similar distribution to $z_i = \mu_\theta(x_i)$, otherwise, k_j cannot receive useful training signal at early training steps. To simplify the initialization, we follow [63] to remove the non-linear function (e.g. ReLU) at the end of μ_θ so that features extracted by μ_θ are approximately Gaussian distributed. With this modification, we can simply initialize k_j with Gaussian distribution.

Cosine-similarity Based Classification Since the model f_ϕ is fine-tuned using the learned dictionary in the inner loop, the quality of the dictionary has a significant impact on final performance. Typical neural network classifiers employ a dot product to compute the logits, where the magnitude and direction could both affect the results. Therefore, the model needs to learn both the magnitude and the direction of k_j. To alleviate this training difficulty, we eliminate the influence of magnitude by using a cosine similarity classifier, where only the direction of k_j can affect the computation of logits. Cosine similarity classifiers have been adopted to replace dot product classifiers for few-shot learning [63, 30] and robust classification [220].

Relationship to kNN Below, we show that Meta-Neighborhoods can be derived as a direct generalization of kNN under a multi-task learning framework. Considering a regression task where the regression target is a scalar, the standard view of kNN is as follows. First, aggregate the k-nearest neighbors of a query $\tilde{x_i}$ from the training set $\mathcal{D}$ as $\mathbf{N}(\tilde{x_i}) = \{(x_j, y_j)\}_{j=1}^{k} \subset \mathcal{D}$. Then, predict an average of the responses in the neighborhood: $\hat{y} = \frac{1}{k}\sum_{j=1}^{k} y_j$.

Instead of simply performing an average of responses in a neighborhood, we frame kNN as a solution to a multi-task learning problem with tasks corresponding to individual neighborhoods as follows. Here, we take each query (test) data $\tilde{x_i}$ as a single task, $\mathcal{T}_i$. To find the optimal estimator on the neighborhood $\mathbf{N}(\tilde{x_i}) = \{(x_j, y_j)\}_{j=1}^k$, we optimize the following loss $\mathcal{L}_{\mathcal{T}_i}(f_i) = \frac{1}{k}\sum_{j=1}^k L(f_i(x_j), y_j)$ where L is a supervised loss, and f_i is the estimator to be optimized. For example, for MSE-based regression, the loss for each task is $\mathcal{L}_{\mathcal{T}_i}(f_i) = \frac{1}{k}\sum_{j=1}^k (f_i(x_j) - y_j)^2$. If one takes f_i to be a constant function $f_i(\eta_j) = C_i$, then the loss is simply $\mathcal{L}_{\mathcal{T}_i}(f_i) = \frac{1}{k}\sum_{j=1}^k (C_i - \zeta_j)^2$, which leads to an optimal $f_i(\tilde{x_i}) = C_i = \frac{1}{k}\sum_{j=1}^k \zeta_j$, the same solution as traditional kNN. Similar observations hold for classification. *Thus, given neighborhood assignments, one can view kNN as solving for individual tasks in the special case of a constant estimator* $f_i(x_j) = C_i$.

With the multi-task formulation of kNN, we can generalize kNN to derive our Meta-Neighborhoods method by considering a non-constant estimator as f_i. For instance, one may take f_i as a parametric output function f_{ϕ_i} (e.g. a linear model or neural networks), and fine-tune the parameter ϕ to ϕ_i for a data x_i according to the loss on neighborhood $\mathbf{N}(x_i)$. Instead of fitting a single label on the neighborhood, a parametric approach attempts to fit a richer (e.g. linear) dependency between input features and labels for the neighborhood. In addition, the multi-task formulation gives rise to a way of constructing meta-learning episodes. Also, we learn both neighborhoods and the function f_ϕ jointly in our Meta-Neighborhoods framework.

2.6 Related Work

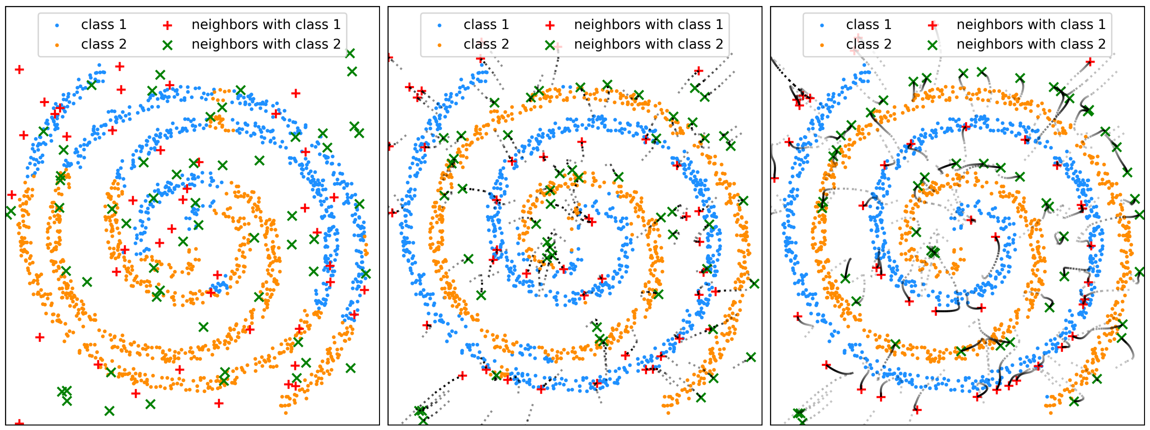
Memory-augmented Neural Networks Augmenting neural networks with memory has been studied in the sentinel work Neural Turing Machine [65], where a neural network can read and write an external memory to record and change its state. Recent works that utilize memory modules generally fall into two categories. One category modifies the memory modules according to hand-crafted rules. For instance, previous works tackling few-shot classifications add a new slot to the memory when the label of a given data does not match the labels of its k-nearest neighbors from the memory [20] or the given data is misclassified [155]. Sprechmann et al. [179] adopt a fixed-size memory that acts as a circular buffer for life-long learning. Another category uses a fully-differentiable memory module and trains it together with neural networks by gradient descent. This type of memory has been explored for knowledge-based reasoning [66], sequential prediction [182] and few-shot learning [87, 166]. Our work also utilizes a differentiable memory but is used to capture the local manifolds and improve the general discriminative learning performance.

Meta-Learning Representative meta-learning algorithms can be roughly categorized into two classes: initialization-based and metric-learning based. Initialization-based methods, such as MAML [56], learn a good initialization for model parameters so that several gradient steps using a limited number of labeled examples can adapt the model to make predictions for new tasks. To further improve flexibility, Meta-SGD [115] learns coordinate-wise inner learning rates, and curvature information is considered in Park & Oliva [146] to transform the gradients in the inner optimization. Metric-learning-based methods focus on using a distance metric on the feature space to compare query set samples with labeled support set samples. Examples include cosine similarity [194] or Euclidean distance [177] to support examples. A learned relation module is employed in [183].

Our model is similar to the initialization-based method: each test sample can be regarded as a new task, and we meta-learn a dictionary that adapts the initial model to a local neighborhood by fine-tuning over queried neighbors. A recent work Meta AuXiliary Learning (MAXL) also explores meta-learning techniques to improve classification performance, where a label generator is meta-learned to generate auxiliary labels so that the auxiliary task trained together with the primary classification task can improve the primary performance.

2.7 Experiments

In this section, we conduct experiments for both classification and regression tasks. To demonstrate the benefits of making predictions in local neighborhoods, we compare to the *vanilla*



(a) Iteration 0 (b) Iteration 800 (c) Iteration 6400

Figure 2.3: Evolution of learnable neighbors and classification results on the test data during training. Two classes are two spirals. Binary predictions for the test set are shown as blue and yellow points. Learnable neighbors are first randomly initialized in (a), then optimized in (b) (c). The dotted black lines are the trajectories of learnable neighbors through training iterations.

model where the same network architecture is used but without the learnable neighborhoods. We also compare to MAXL [122] for classification tasks.

2.7.1 Toy Example: Binary Classification of the Concentric Spiral Dataset

To investigate the behavior of the induced neighbors and how they assist the parametric model to make predictions, we first classify on a 2D toy dataset. In this binary classification task, points from two classes are placed in concentric spirals with a non-linear decision boundary. Although a linear classifier is incapable of capturing this decision boundary, we show that tuning a linear classifier with induced neighbors gives an overall non-linear classifier. In addition, our learned neighborhoods capture the critical manifold structure and concentrate at boundary cases; we also observe semantically relevant learned neighbors in higher dimensions (see Section 2.8.6 and 2.8.7)

In Fig. 2.3, we visualize the evolution of the learned neighbors and the decision boundary. The learnable neighbors (shown in green and red markers for two classes respectively) are first initialized with random keys and values. As we train the model and the neighbors, the learned neighbors are gradually driven to important manifold locations as in Fig. 2.3 (b) and (c). After

Table 2.1: The classification accuracies of our model and the baselines. “MN” denotes Meta-Neighborhoods. Results from three individual runs are reported and the best performance is marked as bold. Given that backbones like ResNet-56 are strong, our consistent improvement is notable.

Datasets	*vanilla*	MAXL	ours		
			vanilla+iFiLM	MN	MN+iFiLM
Backbone: 4-layer ConvNet					
MNIST	99.44±0.03%	99.60±0.02%	99.40±0.02%	**99.62±0.03%**	99.58±0.03%
SVHN	93.02±0.12%	94.06±0.10%	93.95±0.12%	94.46±0.09%	**94.92±0.09%**
MNIST-M	96.18±0.05%	96.85±0.06%	96.99±0.07%	96.55±0.04%	**97.40±0.05%**
PACS	92.55±0.08%	94.85±0.12%	94.45±0.09%	95.19±0.10%	**95.22±0.09%**
Backbone: DenseNet40-BC					
CIFAR-10	94.53±0.10%	94.83±0.09%	94.87±0.08%	95.04±0.11%	**95.22±0.09%**
CIFAR-100	73.92±0.12%	75.64±0.14%	74.66±0.13%	76.32±0.16%	**76.96±0.14%**
CINIC-10	84.92±0.07%	85.42±0.07%	85.11±0.08%	85.73±0.10%	**86.02±0.07%**
Tiny-ImageNet	49.28±0.18%	50.94±0.16%	50.86±0.14%	53.27±0.18%	**54.36±0.15%**
Backbone: ResNet-29					
CIFAR-10	95.06±0.10%	95.31±0.09%	95.17±0.10%	95.56±0.09%	**95.58±0.10%**
CIFAR-100	76.51±0.15%	77.94±0.12%	77.16±0.14%	78.84±0.14%	**79.84±0.11%**
CINIC-10	86.03±0.08%	86.34±0.06%	86.64±0.06%	86.86±0.08%	**87.35±0.09%**
Tiny-ImagNnet	54.82±0.17%	56.29±0.14%	55.59±0.17%	57.36±0.15%	**57.94±0.14%**
Backbone: ResNet-56					
CIFAR-10	95.73±0.08%	96.06±0.07%	96.08±0.08%	96.36±0.07%	**96.40±0.06%**
CIFAR-100	79.64±0.13%	80.36±0.13%	80.04±0.12%	80.58±0.10%	**80.90±0.12%**
CINIC-10	88.21±0.07%	88.30±0.05%	88.57±0.07%	88.61±0.06%	**88.99±0.07%**
Tiny-ImageNet	57.92±0.12%	58.94±0.16%	58.31±0.15%	60.05±0.12%	**60.78±0.13%**
ImageNet	48.41±0.14%	48.83±0.16%	52.03±0.12%	51.85±0.12%	**54.23±0.13%**

training, the linear classifier adapted to local neighborhoods can accurately classify test examples. We use 100 induced neighbors. The labels of neighbors (values in the dictionary) are fixed after random initialization for illustration purposes. The 2D locations of neighbors (keys in the dictionary) are updated with the model. We use negative Euclidean distance as the similarity metric in (2.6) and set T to 0.1.

2.7.2 Image Classification

In this section, we evaluate 9 datasets with different complexities and sizes: MNIST [101], MNIST-M[58], PACS[106], SVHN [64], CIFAR-10 [95], CIFAR-100 [95], CINIC-10 [36], Tiny-ImageNet [163] and ImageNet [40]. Dataset details and preprocessing methods are given in Section 2.8.1.

Our models are compared to two baselines: *vanilla*, a traditional parametric ConvNet with the same architecture as ours but without the learnable dictionary, and MAXL [122], where an auxiliary label generator is meta-learned to enhance the primary classification tasks. For MNIST, MNIST-M, SVHN and PACS, a 4-layer ConvNet is selected as the feature extractor μ_θ. For the other four datasets, three deep convolutional architectures, DenseNet40-BC [80], ResNet29, and ResNet56 [70], are used as μ_θ. f_ϕ is implemented as a cosine-similarity-based classifier with one linear layer for both *vanilla* and our models. Experiment details for our models and baselines are provided in Section 2.8.2.

Results Table 2.1 compares the test accuracy to baselines. We show that both Meta-Neighborhoods (MN) and iFiLM (*vanilla*+iFiLM) can improve over *vanilla*. The best performance is achieved when combining MN and iFiLM (MN+iFiLM), which outperforms the *vanilla* and MAXL baselines across several network architectures and different datasets. This indicates Meta-Neighborhoods and iFiLM are complementary and it is beneficial to adjust both f_ϕ and μ_θ per instance. Our method is also effective at PACS and MNIST-M datasets that contain significant domain shifts.

Note that backbones like ResNet-56 are *already powerful* for these datasets and there is limited room for improvement over the *vanilla* model. For instance, employing ResNet-110 instead of ResNet-56 *only gives 0.14% and 0.40% further improvements* on CIFAR-10 and CIFAR-100, but at the expense of doubling the number of parameters. Yet Meta-Neighborhoods still consistently achieves greater improvements over *vanilla* than the previous SOTA meta-learning method MAXL [122]. Compared to *vanilla* models, Meta-Neighborhoods with the same backbone architecture contains extra trainable parameters stored in the dictionary M. However, as discussed in Section 2.8.3, the performance boost in our paper originates from adjusting models using neighbors, rather than a naive increase in the number of parameters.

Since we implement f_ϕ as a cosine similarity classification layer, ϕ can be regarded as the prototypes for each class. To verify that fine-tuning over neighborhoods helps with the classification, we compare the cosine similarity between the extracted feature z_i and its corresponding ground-truth prototypes $\phi[y_i]$ before and after fine-tuning, where y_i is the class label for z_i. From

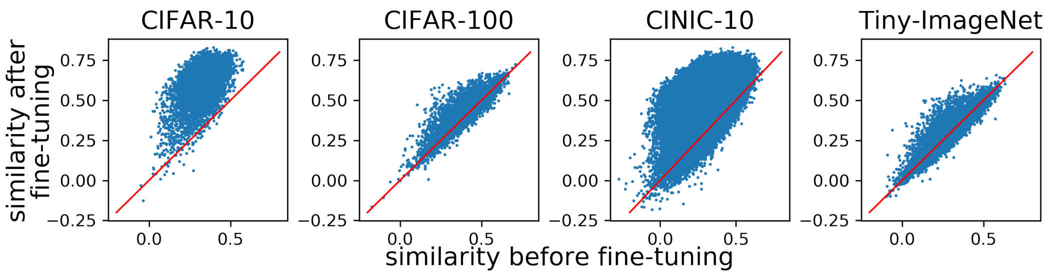


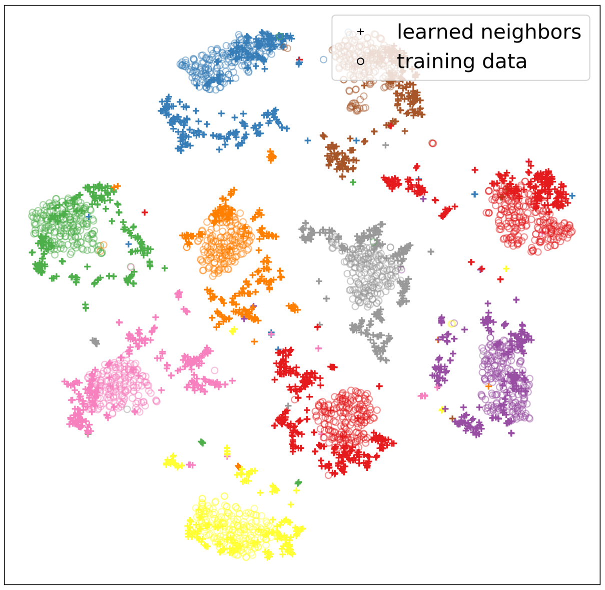
Figure 2.4: Cosine similarities between features and their corresponding ground-truth class prototypes. Each blue point denotes a testing sample. We expect most samples to be located above the red lines, meaning larger similarities after fine-tuning.

Fig. 2.4, we can see that the cosine similarities increase after fine-tuning for most test examples, which indicates better predictions after fine-tuning.

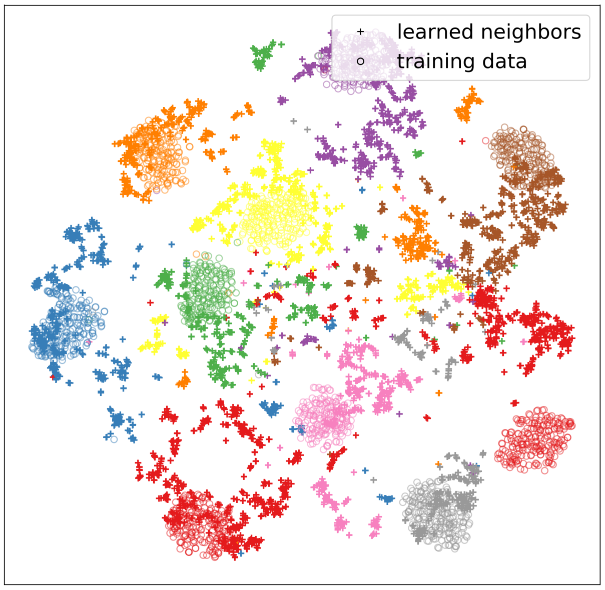
We discuss the inference speed of our model in Section 2.8.5.

Analysis of Learned Neighbors In Fig. 2.5, we use t-SNE [135] to visualize the 2D embeddings of the learned neighbors (marked as "+") along with the training data (marked as "o") on CIFAR-10 and MNIST. The 2D embeddings of training data are computed on the features $z_i = \mu_\theta(x_i)$, and embeddings of learned neighbors are computed on keys k_j. Classes of the learned neighbors are inferred from values v_j. It is shown that our model learns neighbors beyond the training set as the learned neighbors do not completely overlap with the training data, and the learned neighbors represent "hard cases" around class boundaries to assist our model in making better predictions. It is also interesting to note that this follows the same trend as our toy example in Fig. 2.3.

We further investigate whether the learned neighbors are semantically meaningful by retrieving their 5-nearest neighbors from the test set. As shown in Section 2.8.6, the retrieved 5-nearest neighbors for each learned neighbor not only come from the same class but also represent a specific sub-category concept. In Section 2.8.7, we quantitatively show that our method has superior sub-category discovery performance than *vanilla* on CIFAR-100: our method achieves 63.3% accuracy on the 100 fine-grained classes while *vanilla* only achieves 59.28%. This indicates our learned neighbors can preserve fine-grained details that are not explicitly given in the supervision signal. Qualitative results are shown in Fig. 2.6.



(a) MNIST



(b) CIFAR-10

Figure 2.5: t-SNE visualization of the learned neighbors and training data on MNIST and CIFAR-10. Learned neighbors are marked as “+” and real training data are marked as “o”. The class information is represented by colors. Please zoom in to see the differences between “+” and “o”.

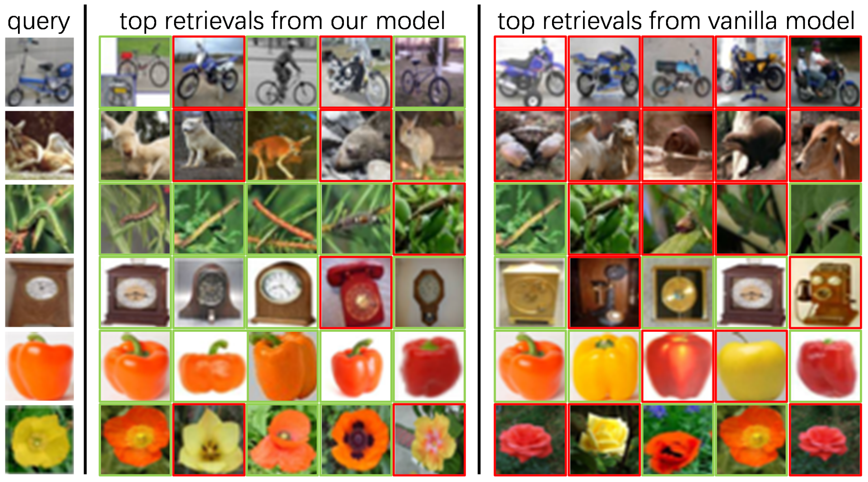

query
top retrievals from our model
top retrievals from vanilla model

Figure 2.6: Sub-category image retrieval quality of our model and the *vanilla* model. Correct retrievals have green outlines and wrong retrievals have red outlines.

Table 2.2: Test MSE of our model, kNN and the *vanilla* baseline on five datasets. n and d respectively denote the dataset size and the data dimension.

Datasets	n	d	kNN	*vanilla*	Meta-Neighborhoods
music	515345	90	0.6812±0.0062	0.6236±0.0056	**0.6088±0.0050**
toms	28179	96	0.0602±0.0083	0.0594±0.0080	**0.0531±0.0073**
cte	53500	384	0.00134±0.00023	0.00121±0.00022	**0.00109±0.00015**
super	21263	80	0.1126±0.0061	0.1132±0.0060	**0.1077±0.0068**
gom	1059	116	0.5982±0.0521	0.5949±0.0515	**0.5681±0.0563**

2.7.3 Regression

We use five publicly available datasets with various sizes from UCI Machine Learning Repository [43]. For regression tasks, we found learning neighbors in the input space yields better performance compared to learning neighbors in the feature space. As a result, the feature extractor μ_θ is implemented as an identity mapping function. We compare our model to kNN and *vanilla* baseline using the mean square error (MSE). The *vanilla* baseline is a multilayer perceptron for regression. We searched for the best network configuration for the *vanilla* model on every dataset by varying the number of layers in $\{2, 3, 4, 5\}$ and the number of neurons at each layer in $\{32, 64, 128, 256\}$. For each dataset, our model uses the same network architecture to *vanilla*. Model details, training details, and hyperparameter settings are given in Section 2.8.8. 5-fold cross-validation is used to report the results in Table 2.2. Our model has lower MSE scores compared to the *vanilla* model across the five datasets. The results of Meta-Neighborhoods and *vanilla* are statistically different based on paired Student's t-test with a significance of 0.05. We found that naively increasing the model complexity for *vanilla* baseline can not further improve its performance due to over-fitting, but our method can as it takes advantage of non-parametric neighbor information.

2.8 Experiment Details for Classification

2.8.1 Dataset Details

All datasets except ImageNet were resized to resolution 32×32 to facilitate fast experimentation. ImageNet was resized to resolution 64×64. CIFAR-10/100 are image classification datasets containing a training set of 50K and a testing set of 10K 32×32 color images across the 10/100 classes. CINIC-10 has 270K 32×32 images across 10 classes equally split into three parts for training, validation, and testing. Tiny-ImageNet has a training set of 100K and a testing set of 10K 64×64 images across the 200 classes. PACS contains domain shifts (Art painting, Cartoon, Photo and Sketch), and we train on all domains to test our model's ability to fit on all domains. PACS is split as 9K/1K/10K for training/validation/testing. MNIST-M also contains domain shifts due to the random background. MNIST-M is split as 55K/5K/10K for training/validation/testing.

2.8.2 Implementation Details

We find adaptive learning rate optimizers like Adam [92] are more effective than SGD at optimizing the dictionary, as these optimizers can compute individual adaptive learning rates for different parameters. In all the experiments, we adopt AdamW [124], a variant of Adam [92] with a correct weight decay mechanism and better generalization capability.

Our models are trained by AdamW with a weight decay rate of 7.5e-5, an initial learning rate of 1e-3 and a batch size of 128. For CIFAR-10 and CIFAR-100, we train for 400 epochs with a learning rate reduced to 1e-4 at epoch 300. For CINIC-10 and Tiny-ImageNet, models are trained for 350 epochs with the learning rate reduced to 1e-4 at epoch 250. We train *vanilla* baselines by AdamW in the same way as our models.

We initialize k_j and v_j with Gaussian distribution $\mathcal{N}(0,\, 0.01)$, and apply a softmax over v_j to make it a probability distribution over multiple classes. Cosine similarity is adopted as the similarity metric in (2.6). We set T in (2.6) to 0.2. For CIFAR-10 and CINIC-10, the number of dictionary entries S is set to 5000, and for CIFAR-100 and Tiny-Imagenet S is set to 10000 in the main results. For the results in Table 2.1, we set the number of inner loop steps to 1 and set α to be a learnable scalar.

When optimizing the dictionary at the early stage, some entries might always receive larger gradients than others due to the random initialization, which causes the model to exploit only part of the dictionary entries. To boost the diversity of used entries, we randomly dropout 50% of the entries during training. This also helps to prevent overfitting and improve training speed.

For models (Densenet40-BC, ResNet) with iFiLM, we learn a learnable fix-sized dictionary $M_l = \{k_j^l, \gamma_j^l, \beta_j^l\}_{j=1}^{S_l}$ with every batch normalization layer, where S_l is set to 10 for all M_l.

We ran MAXL using code provided in [122] and performed extensive hyperparameter tuning to report the best results. We run MAXL with SGD of a learning rate of 0.1 with momentum 0.9 and weight decay 5×10^4, and hierarchies to set to 5 as recommended in MAXL. We also try a larger learning rate of 0.01. We dropped the learning rate by half for every 50 epochs with a total of 200 epochs.

2.8.3 *vanilla* Models with Extra Parameters

Compared to *vanilla* models, our best model (MN+iFiLM) with the same backbone architectures contain $S \times (d + C)$ extra trainable parameters in the dictionary M and $3 \times S^l \times C^l$ extra trainable parameters in M^l, where S is the number of dictionary entries in M, d is the dimension of the feature vector, C is the number of classes, S^l is the number of dictionary entries in M^l and C^l is the number of feature map channels. Since we set a small $S^l = 10$, extra parameters in M^l is negligible compared to the remaining model parameters. Therefore we only consider the extra parameters in M in this section.

To investigate whether *vanilla* models can also benefit from more parameters as our model, we add an extra fully connected layer with the same number of extra parameters to *vanilla* models. According to Fig. 2.7, *vanilla* models with extra parameters have inferior validation accuracy than *vanilla* models without extra parameters across three backbone architectures on CIFAR-

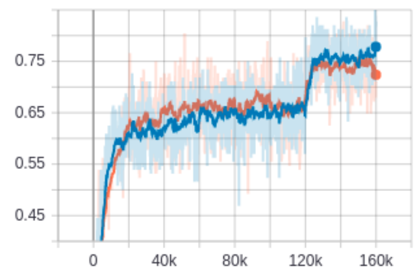
100. The same observation holds for other datasets. Therefore, *vanilla* models can not benefit from more parameters as our model, and the performance boost in our paper originates from fine-tuning using neighbors, rather than a naive increase in parameters.

2.8.4 Ablation Study of S and T

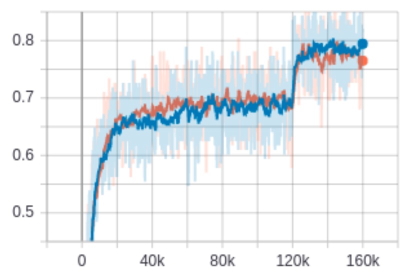
We investigate the impact of hyperparameters, S and T on CIFAR-100 using ResNet-29. As shown in Fig. 2.8, the testing accuracy increases with the increase of the number of dictionary entries S (with T set to 0.2), which indicates a better fine-tuning of ϕ with the help of more learnable neighbors. The temperature T controls the "peakiness" of the similarity distribution in (2.6). It is set to a fixed value rather than learned in all experiments. If we enable T to be learnable, it always grows to a small value, which makes the model only pay attention to a small number of entries and leads to over-fitting. On the other hand, if T is too large, the model will pay uniform attention to all entries in M, which leads to under-fitting. According to Fig. 2.8, the optimal range of $\frac{1}{T}$ is [3,7] (with S set to 10000).

We show in Table 2.1 that our method consistently improves performance regardless of the choice of backbones, indicating our method can further improve over an even bigger model.

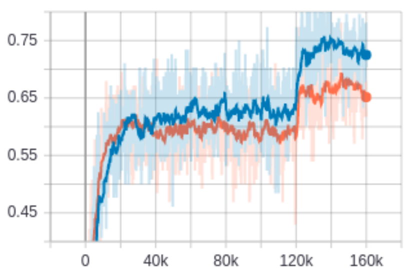
Note that our method is particularly helpful for low-capacity models, which usually handle simpler tasks like MNIST/SVHN classification and regression. In the regression experiment in Section 2.7.3, we show that naively increasing the model capacity can't effectively further



(a) ResNet-29



(b) ResNet-56



(c) DenseNet40-BC

Figure 2.7: Validation accuracies of *vanilla* models with (shown in orange) and without (shown in blue) extra parameters on CIFAR-100 across three backbone architectures.

improve performance on these simple tasks due to over-fitting, but our method can as it takes advantage of non-parametric neighbor information.

2.8.5 Inference Speed

Due to the neighbor searching process and fine-tuning process, our method is slower at the testing time compared to the *vanilla* testing process which only requires a single feed-forward propagation. However, our method is only approximately 2 times slower than the vanilla models due to the following reasons: (1) our searching space is small (only 5000 neighboring points) (2) the attention calculation and the fine-tuning step are parallelized efficiently across multiple GPU threads (3) for the classification task, we only fine-tune the parameters of the classification layer rather than the whole model (4) we only fine-tune for a small number of steps (1 or 3).

2.8.6 Visualizing Learned Neighbors by Retrieving Real Neighbors

We further investigate whether the learned neighbors are semantically meaningful by retrieving their 5-nearest neighbors from the test set.

Examples on CIFAR-10 and MNIST are respectively shown in Fig. 2.9 and 2.10. We found most entries can retrieve consistent neighbors. It is shown that the retrieved 5-nearest neighbors for each learned neighbor not only come from the same class but also represent a specific sub-category concept. For instance, both of the entries on the fifth row of Fig. 2.9 represent “ship”, but the first represents “steamship” while the second represents “speedboat”.

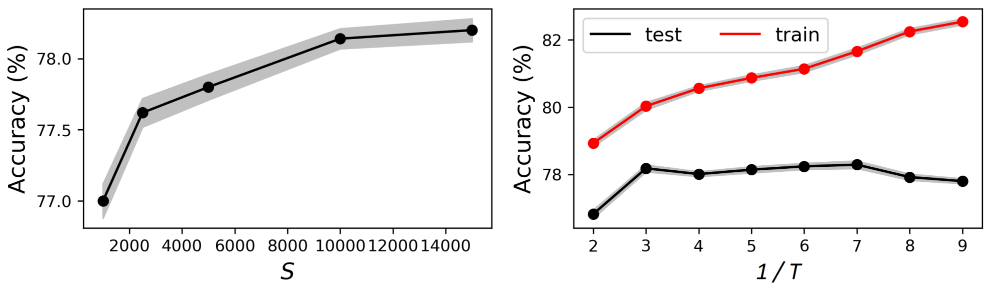


Figure 2.8: Ablation studies of S and T on CIFAR-100.

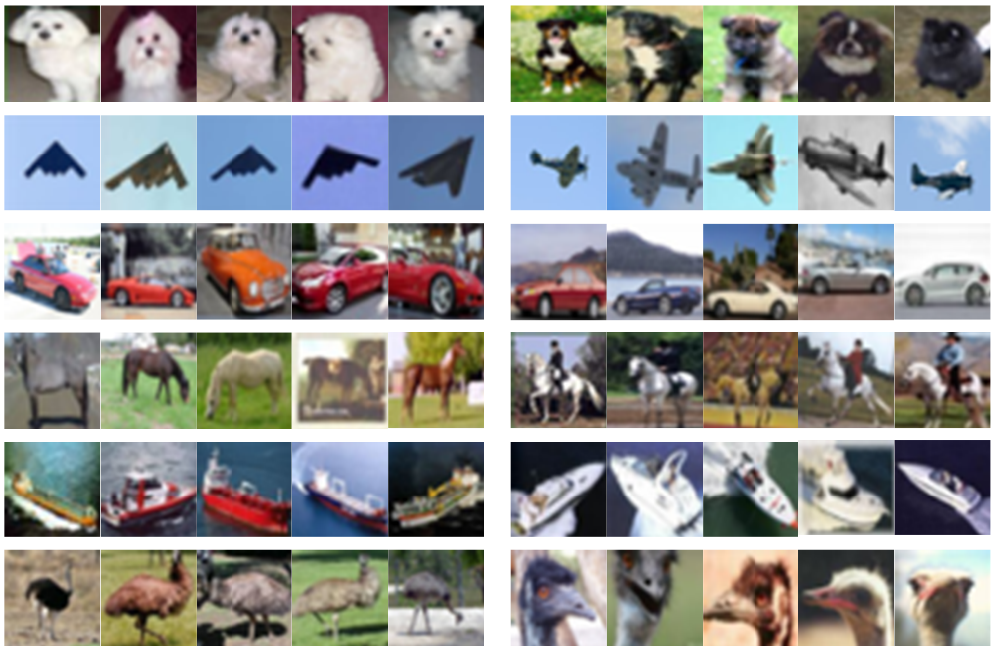


Figure 2.9: 5-nearest neighbors of 12 dictionary entries retrieved using k_j in (2.4) from the CIFAR-10 test set. Entry indexes and entry classes inferred from v_j are shown on the left of each group of images. By comparing the two entries on the same row, we discover that different entries represent different fine-grained sub-category concepts.

2.8.7 Sub-category Discovery

To quantitatively measure the sub-category discovery performance, we train our model (S is set to 5000) and the *vanilla* cos-adamw model with the same Densenet40 backbone on CIFAR-100 only using its coarse-grained annotations (20 classes), and evaluate the classification accuracy on the fine-grained 100 categories using kNN classifiers, which is called induction accuracy as in Huh et al. [81]. For the *vanilla* model, the kNN classifier uses the feature $z_i = \mu_\theta(x_i)$, while for our model, the classifier uses the attention vector $\vec{\omega} = [\omega(z_i, k_1), \omega(z_i, k_2), ..., \omega(z_i, k_S)]$ over all entries in M. Our kNN and *vanilla*'s kNN achieve similar accuracy on the coarse 20 classes (80.18% versus 80.30%). However, on the fine-grained 100 classes, our kNN achieves 63.3% while the *vanilla*'s kNN only achieves 59.28%. This indicates our learned neighbors can preserve fine-grained details that are not explicitly given in the supervision signal. Examples of nearest neighbors retrieved are shown in Fig. 2.6.

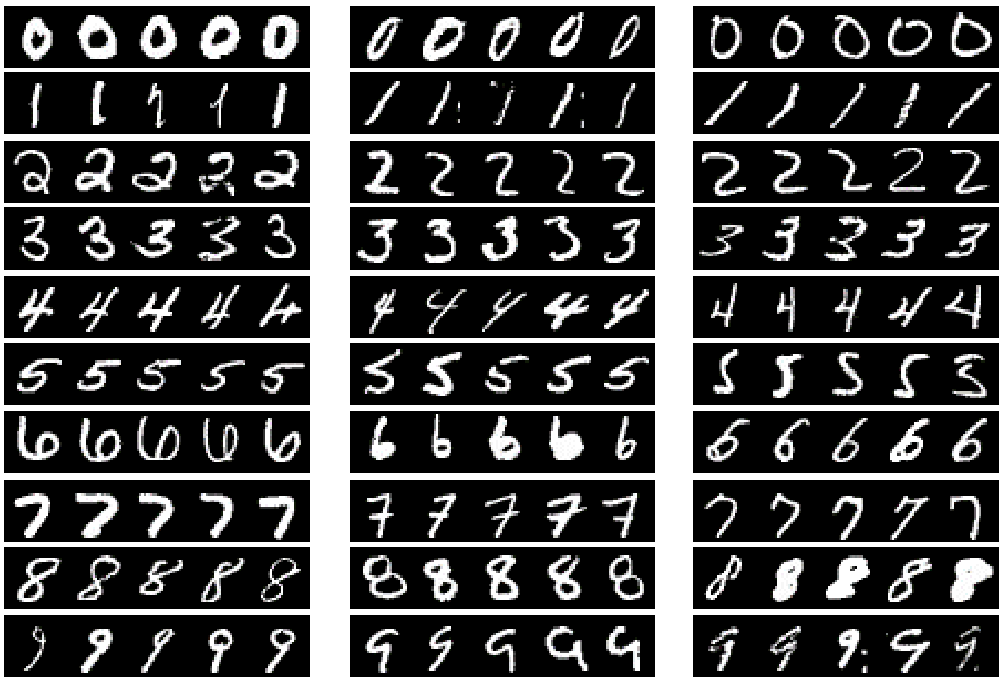


Figure 2.10: 5-nearest neighbors of 30 dictionary entries retrieved using k_j in (2.4) from the MNIST test set. By comparing the three entries on the same row, we discover that different entries represent different fine-grained attributes such as stroke widths, character orientations and fonts.

2.8.8 Experiment Details for Regression

We use five publicly available datasets with various sizes from UCI Machine Learning Repository: *music* (prediction of the release year of a song from audio features), *toms*, *cte* (Relative location of CT slices on axial axis), *super* (Superconduct), and *gom* (Geographical Original of Music). All datasets are normalized dimension-wise to have zero means and unit variances.

For regression tasks, we found learning neighbors in the input space yields better performance compared to learning neighbors in the feature space. As a result, our model for regression only consists of an output network f_ϕ and a dictionary M. It is trained with the loss in (2.3). A learning rate of 1e-3 and a batch size of 128 are used, and the best weight decay rate is chosen for each dataset. The training stops if the validation loss does not reduce for 10 epochs. We initialize k_j with Gaussian distribution $\mathcal{N}(0,\, 0.01)$ and v_j with uniform distribution in the same range of the regression labels. Cosine similarity is adopted to implement the similarity metric in (2.6). We use 1000 dictionary entries and set T to 0.1 based on the validation performance. Because there is no batch normalization layer in f_ϕ, iFiLM is not used in this regression experiment.

2.9 Closing Remarks

In this chapter, we introduced Meta-Neighborhoods, a novel meta-learning framework that adjusts predictions based on learnable neighbors (related instances). It is interesting to note that in addition to directly generalizing kNN, Meta-Neighborhoods provides a learning paradigm that aligns more closely with human learning. Human learning jointly leverages previous examples both to shape the perceptual features we focus on and to pull relevant memories when faced with novel scenarios [97]. In much the same way, Meta-Neighborhoods use feature-based models that are then adjusted by pulling memories from previous data. We show through extensive empirical studies that Meta-Neighborhoods improve the performance of already strong backbone networks like DenseNet and ResNet on several benchmark datasets. In addition to providing a greater gain in performance than previous state-of-the-art meta-learning methods like MAXL, Meta-Neighborhoods also works both for regression and classification, and provides further interpretability as shown by Fig. 2.9 and 2.10.

Compared to vanilla neural networks and the previous work MAXL, Meta-Neighborhoods can explicitly find related instances to provide some understanding of the decision-making process and use them to improve the model performance. However, the limitation of Meta-Neighborhoods is that it is approximately 3 times slower to train and 2 times slower at inference compared to vanilla neural networks. This slowdown is caused by the related instances retrieval step and the fine-tuning step.

CHAPTER 3: LEVERAGING RELATED NEIGHBOR WAVEFORMS FOR IMPROVED AUDIO SYNTHESIS

3.1 Notation

- $k_i \in \mathbb{R}^L$: a one-dimensional waveform of length L. $k_i[j]$ is the value at the j-th position of k_i.
- $D = \{k_i\}_i^N$: a learnable dictionary containing N waveforms.
- $\omega_1(t), \omega_2(t), ...\omega_N(t)$: time-varying attention weights over N wavetables at timestep t. $\sum_{i=1}^N \omega_i(t) = 1$ and $\omega_i(t) \geq 0$.
- $f_0(t)$: fundamental frequency at timestep t. The fundamental frequency, often referred to simply as the fundamental, is defined as the lowest frequency of a periodic waveform. In music, the fundamental is the musical pitch of a note that is perceived as the lowest partial present.

3.2 Introduction

In the previous chapter, we show that leveraging related neighbor points can improve discriminative learning performance on image classification and tabular data regression tasks. In this chapter, we show that leveraging related neighbor waveforms can also help improve audio synthesis performance.

Although machine learning has revolutionized modern audio classification [172] and synthesis [48] with unprecedented progress; fast, robust, and real-time neural audio synthesis remains a challenge. Most neural synthesis models generate waveforms directly in the time domain, or from their corresponding Fourier coefficients in the frequency domain [44]. However, both approaches have their drawbacks. Fourier-based models–such as GANSynth [48] suffer from the phase-alignment problem, as the Short-time Fourier Transform (STFT) is a representation over windowed wave packets. Autoregressive waveform models, such as WaveNet [189], avoid these issues by generating the waveform a single sample at a time. They are not constrained by the bias over generating wave packets and can express arbitrary waveforms. However, they require larger and more data-hungry networks, as they do not take advantage of a bias over oscillation. In addition, the use of teacher-forcing during training leads to exposure bias during generation, where errors with feedback can compound. Also, their generation speed is slow as they only generate autoregressively.

One of the causes of the above-mentioned problems is that these methods fail to take advantage of the nature of harmonic audio. Many acoustic oscillators, such as the human voice or a bowed violin string, produce complex tones that are more or less periodic and thus are composed of harmonic components with frequencies that are near matches to integer multiples of the fundamental frequency. Also, real-world objects that generate sound often exhibit physics that are well described by harmonic oscillations. These include vibrating strings, membranes, hollow pipes and human vocal chords [176]. Based on this interesting phenomenon, DDSP [44] proposes a novel approach that learns to combine several sinusoidal waveforms with integer multiples of the fundamental frequency. DDSP describes a family of techniques utilizing strong inductive biases from DSP combined with modern ML. Successful applications include audio synthesis [45], pitch detection [46], artificial reverberations [104], IIR filters [13] and audio effect manipualtion [157]. However, DDSP uses a fixed set of 100 sinusoidal waveforms that are not customized for different datasets. Thus, it is redundant to use the same set of waveforms for all the datasets.

In this paper, inspired by compressive sensing techniques [57], we take an alternative approach to learn a set of related neighbor waveforms that are more compact and customized toward the specific dataset we want to train. To synthesize a new waveform, we train a neural network to find and combine related neighbor waveforms for high-quality, real-time audio synthesis. We call our approach Differentiable Wavetable Synthesis (DWTS) as our approach is inspired by Wavetable Synthesis (WTS) that is well-suited to the real-time synthesis of periodic and quasi-periodic signals. We demonstrate three key contributions to neural audio synthesis using this approach: high fidelity synthesis, wavetables transferrable to other tasks and an order of magnitude reduction in computational complexity.

3.3 Problem Formulation

Given that monophonic audio is composed of multiple cycles, the core insight of this paper is that every cycle can be synthesized by linearly combining a dictionary of one-period waveforms. We refer to such a dictionary as a wavetable and define it as $D = \{k_i\}_i^N$ where N is the number of waveforms and $k_i \in \mathbb{R}^L$ denotes a one-period waveform of length L. Unlike the traditional wavetable synthesis technique [15] that uses hand-crafted waveforms, we learn D using gradient descent jointly with other components in our model during training. During inference, D is frozen and our model only needs to compute linear combination weights to combine the waveforms in D to synthesize different audio. In this light, the waveforms in D can be conceptually regarded as a set of basis vectors that spans a L-dimensional vector space.

3.4 Background

Our work is built on Differentiable Digital Signal Processing (DDSP) [44], which enables direct integration of classic signal processing elements with deep learning methods. Despite significant advances in neural audio synthesis, such as methods based on CNN, GAN, and autoregressive models, these methods suffer from their own drawbacks such as wrong phase alignment, spectral leakage, slow synthesis speed and model collapse. According to the DDSP, the main reason that causes these drawbacks is that these methods fail to utilize existing knowledge of how sound is generated and perceived. Unlike deep learning based methods, a class of traditional approaches (vocoders/synthesizers) successfully consider strong domain knowledge of signal

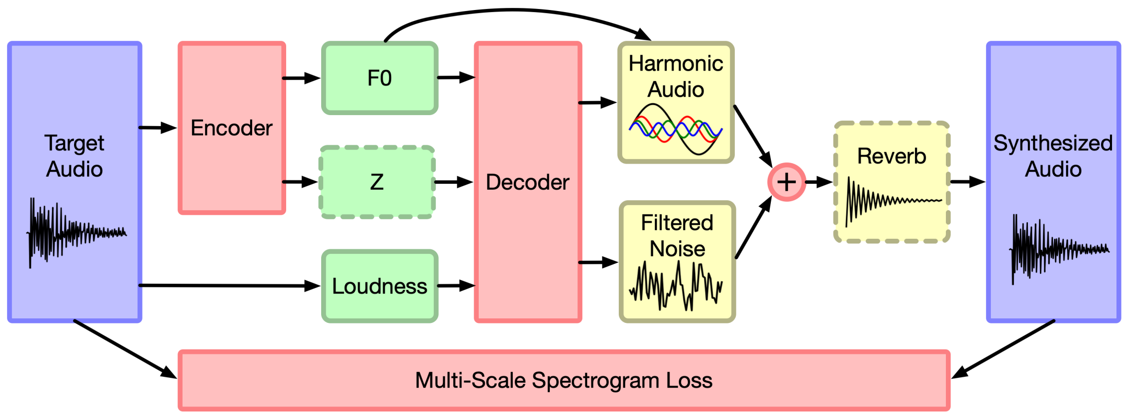


Figure 3.1: Architeture of DDSP [44]. Figure taken from [44].

processing and perception. But these methods are less expressive and seem incompatible with deep learning approaches.

DDSP is the first end-to-end differentiable method that combines the strong inductive bias of signal processing with deep learning approaches. DDSP can generate high-fidelity audio without autoregressive models, large neural networks, or adversarial losses.

The architecture of DDSP is shown in Fig. 3.1, where an autoencoder is used to learn a disentangled and interpretable latent space for audio resynthesis. Given an input audio, the encoder extracts time-varying fundamental frequency f_0 and a latent vector z that contains information about input timber. The time-varying loudness information is also extracted by a hand-engineered algorithm. Based on this human-understandable and disentangled information, a decoder predicts how to combine harmonics components and noise components for reconstructing the input audio. The overall model is end-to-end differentiable and trained using a multi-scale spectrogram loss, which is better than the point-wise loss on the raw waveform as two perceptually identical audio samples may have distinct waveforms, and point-wise similar waveforms may sound very different. Multi-scale loss can better compare two waveforms across multiple spatial-temporal resolutions. Given this specifically designed architecture, DDSP is highly interpretable and thus enables manipulation of individual components to independently control attributes of the generated audio, such as pitch, loudness, extrapolation to unseen pitches and timbre transfer.

The sinusoidal oscillator [10] is the heart of DDSP. A bank of oscillators that outputs a signal $x(n)$ over discrete time steps, n, can be expressed as:

$$x(n) = \sum_{k=1}^{K} A_k(n) \sin(\phi_k(n)), \tag{3.1}$$

where $A_k(n)$ is the time-varying amplitude of the k-th sinusoidal component and $\phi_k(n)$ is its instantaneous phase. The phase $\phi_k(n)$ is obtained by integrating the instantaneous frequency $f_k(n)$:

$$\phi_k(n) = 2\pi \sum_{m=0}^{n} f_k(m) + \phi_{0,k}, \tag{3.2}$$

where $\phi_{0,k}$ is the initial phase that can be randomized, fixed, or learned.

For a harmonic oscillator, all the sinusoidal frequencies are harmonic (integer) multiples of a fundamental frequency, $f_0(n)$, i.e., $f_k(n) = kf_0(n)$. The fundamental frequency, often referred to simply as the fundamental, is defined as the lowest frequency of a periodic waveform. In music, the fundamental is the musical pitch of a note that is perceived as the lowest partial present. Thus the output of the harmonic oscillator is entirely parameterized by the time-varying fundamental frequency $f_0(n)$ and harmonic amplitudes $A_k(n)$. To aid interpretability DDSP further factorizes the harmonic amplitudes:

$$A_k(n) = A(n)c_k(n). \tag{3.3}$$

into a global amplitude $A(n)$ that controls the loudness and a normalized distribution over harmonics $c(n)$ that determines spectral variations, where $\sum_{k=0}^{K} c_k(n) = 1$ and $c_k(n) \geq 0$. DDSP also constrains both amplitudes and harmonic distribution components to be positive through the use of a modified sigmoid nonlinearity.

3.5 Methods

Similar to Meta-Neighborhoods in Chapter 2 that learns a dictionary of related instances to assist predictions, we propose an approach Differentiable Wavetable Synthesis (DWTS) [169] that learns a dictionary of related basic instances that can be combined to generate authentic audios. Our model has a similar architecture shown in Fig. 3.1, except that we avoid using the sinusoidal oscillator in DDSP. Instead, we take a wavetable synthesis approach. This idea to synthesize audio from a set of basic elements can date back to the 1970s when a sound synthesis technique called Wavetable Synthesis [76] was introduced to create periodic waveforms. This technique synthesizes periodic signals by combining a table of hand-crafted waveforms, while our proposed method DWTS learns such a dictionary of waveforms (related instances) that best synthesize a target dataset.

3.5.1 Wavetable as learnable dictionary

We define a learnable dictionary $D = \{k_i\}_i^N$ where N is the number of wavetables and $k_i \in \mathbb{R}^L$ denotes a one-period wavetable of length L. When a wavetable begins and ends on different values, this discontinuity causes synthesis artifacts. We append $k_i[L+1]$ to k_i and set $k_i[L+1] = k_i[0]$. A wavetable k_i now contains $L+1$ elements with L learnable parameters. During training, we learn D using gradient descent jointly with other parameters. During inference, D is frozen and treated as a traditional, static collection of wavetables.

3.5.2 Time-varying attention

By sequential morphing between wavetables, timbre can be changed over time [77]. Inspired by [75] we generalize morphing as a time-varying linear attention over all wavetables i.e. $\omega_1^N, \omega_2^N ... \omega_T^N$ where N and T are numbers of wavetables and timesteps respectively with constraints $\sum_{i=1}^N \omega_i(n) = 1$ and $\omega_i(n) \geq 0$. The time-varying linear attention is predicted by the encoder in Fig. 3.1.

3.5.3 Phase

Our model "draws" wavetables directly in the time domain. Phase relationships within and across wavetables can be controlled without needing to coherently manage independent magni-

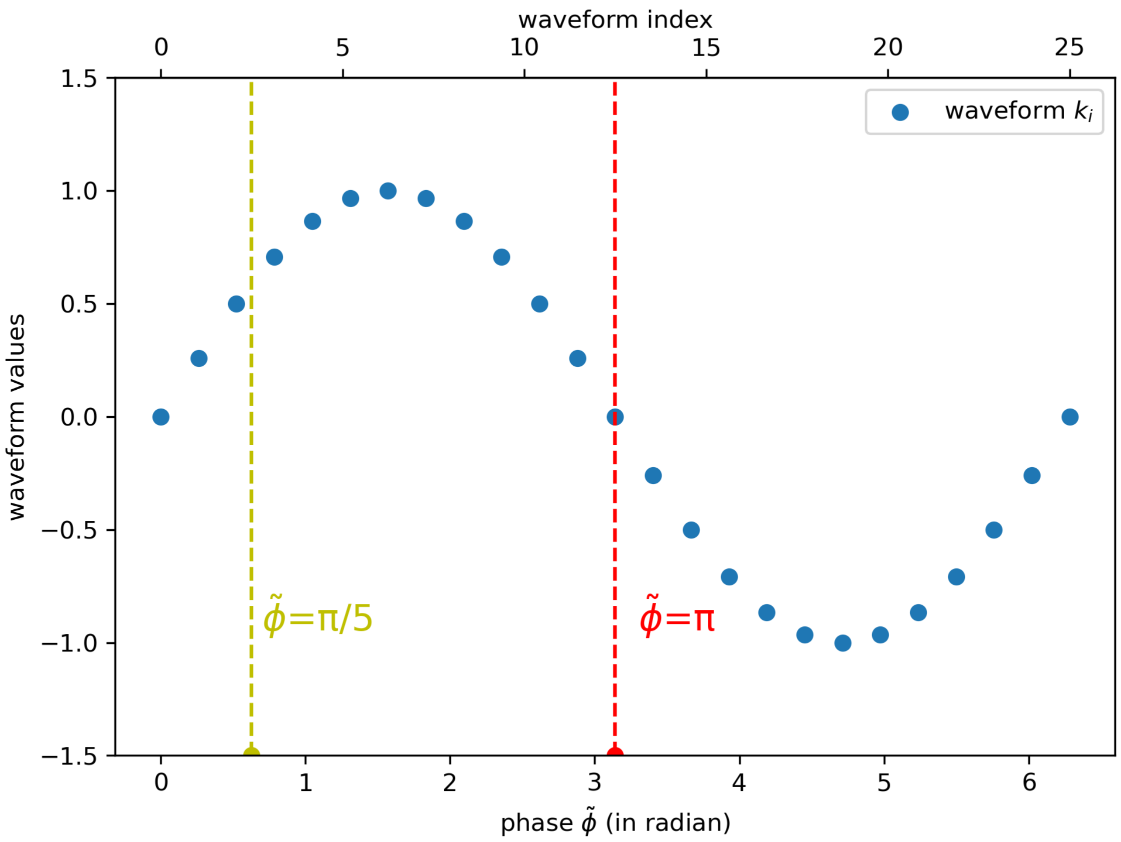


Figure 3.2: Illustration of how we use the instantaneous modulo phase $\tilde{\phi}(n)$ to retrieve values from a waveform k_i. In this example, we have a waveform k_i with a length of 25 in a sinusoidal shape. Note that k_i only has values defined at the integer indices 0, 1, ..., 24. When $\tilde{\phi} = \pi$, we can retrieve a value of 0 at the integer index 13 of k_i (as shown by the red vertical dashed line). However, when $\tilde{\phi} = \frac{\pi}{5}$, which corresponds to a non-integer index of k_i (as shown by the yellow vertical dashed line), we need to use some interpolation techniques to retrieve the value.

tudes and phases in the complex frequency domain. This contrasts with [45], where the initial phases of all harmonic components are fixed at 0.

3.5.4 Synthesis

At the heart of WTS is a phase accumulator [15]. Given an input sequence of time-varying fundamental frequency $f_0(n)$ over discrete time steps n, we can compute the instantaneous phase ϕ by integrating $f_0(n)$:

$$\phi(n) = 2\pi \sum_{m=0}^{n} f_0(m). \tag{3.4}$$

As illustrated by Fig. 3.2 and will be introduced below, we use the phase information $\phi(n)$ to retrieve values in the waveform k_i to synthesize the output signal x. However, k_i is a one-period waveform, which means that we need to reuse the same waveform k_i on all the periods determined by the phase information $\phi(n)$ (a period finishes whenever $\phi(n)$ reaches 2π). For this purpose, we need to compute the instantaneous modulo phase $\tilde{\phi}(n)$:

$$\tilde{\phi}(n) = \phi(n) \text{ mod } 2\pi, \tag{3.5}$$

where mod denotes a module operation. Now, we are ready to use $\tilde{\phi}(n)$ to retrieve values from k_i. However, the range of $\tilde{\phi}(n)$ is $[0, 2\pi]$ while the length of k_i is L. To fix this incompatibility, $\tilde{\phi}(n)$ is normalized into a fractional index $\tilde{j}(n) = \frac{L}{2\pi}\tilde{\phi}(n) \in [0, L]$. We synthesize the signal $x(n)$ by linearly combining wavetables k_i in D via:

$$x(n) = A(n) \sum_{i=1}^{N} \omega_i(n) \cdot \Phi(k_i, \tilde{j}(n), \kappa), \tag{3.6}$$

where $A(n)$ is a time-varying amplitude controlling the signal's overall amplitude and ω_i denotes the time-varying attention on k_i. $A(n)$ and $\omega_i(n)$ are constrained positive via a sigmoid. The function $\Phi(k_i, \tilde{j}(n), \kappa)$ is an operator that retrieves the $\tilde{j}(n)$-th element of the vector k_i by using an interpolation kernel κ to approximate $k_i[\tilde{j}(n)]$ when $\tilde{j}(n)$ is a non-integer (illustrated by the yellow dashed line in Fig. 3.2):

$$\Phi(k_i, \tilde{j}(n), \kappa) = \sum_{m=1}^{L+1} k_i[m] \cdot \kappa(\tilde{j}(n) + 1 - m). \tag{3.7}$$

Whilst more sophisticated interpolation kernels exist (cubic, spline, etc.), we use linear interpolation for optimal real-time performance. When κ is a linear interpolation kernel, only two consecutive samples of the original waveform k_i are involved in the computation of an interpolated sample and $\Phi(k_i, \tilde{j}(n), \kappa)$ is

$$\Phi(k_i, \tilde{j}(n), \kappa) = \sum_{m=1}^{L+1} k_i[m] \cdot \max(0, 1 - |\tilde{j}(n) + 1 - m|). \tag{3.8}$$

3.5.5 Initialization

k_i is randomly initialized with a zero-centered Gaussian distribution $\mathcal{N}(0, \sigma^2)$. We empirically find using a small $\sigma = 0.01$ improves training.

3.5.6 Anti-aliasing

At high f_0, the frequencies of upper harmonics in a wavetable can be above Nyquist ((i.e. $0.5 \times f_s$ where f_s denotes the sampling rate)) and must be removed before lookup to prevent aliasing. Therefore, we need to limit the number of harmonics kept in k_i to be smaller than the upper bound $K = \lfloor 0.5 \times f_s / f_0(n) \rfloor$ [77]. We first convert k_i to the frequency domain via the discrete Fourier transform (DFT) as $\mathbf{W}_i = DFT(k_i)$. Then only the first K elements of $\mathbf{W}_i$ are kept while the subsequent elements are set to 0 as $\hat{\mathbf{W}}_i = \mathbf{m} \odot \mathbf{W}_i$ where $\mathbf{m}_j = 1$ if $j \leq K$ otherwise $\mathbf{m}_j = 0$ and $\odot$ denotes the Hadamard product. Lastly, we convert $\hat{\mathbf{W}}_i$ back to the time domain via the inverse DFT (IDFT) as $\hat{k}_i = IDFT(\hat{\mathbf{W}}_i)$. Given that the Fourier transform and the inverse Fourier transform have the linearity property and (3.6) is a linear combination of the waveforms in D, in practice, we directly apply the band limit operation to the synthesized signal $x(n)$ rather than apply it individually N times for every k_i to increase the computation speed. Since all the computations of this anti-aliasing step are differentiable, it can be included in our model for end-to-end training.

This filter also prevents high-frequency components present in the Gaussian noise initialization from causing aliasing at the start of training. Without this frequency-dependent anti-aliasing filter, we found aliasing artifacts alone prevented meaningful learning.

3.5.7 Model

We adopt an identical input tuple $(f_0(n), l(n), z(n))$ [45]. Fundamental frequency $f_0(n)$ in (3.4) is extracted by a pretrained CREPE model [91] with fixed weights. Loudness $l(n)$ is an A-weighted log-magnitude extracted deterministically from audio. The residual embedding $z(n)$ is extracted from MFCC's via an encoder. For direct comparison, we use identical encoder and decoder architectures with approximately 7M parameters in total.

Unlike DDSP which only has learnable parameters in the encoder and the decoder, our model also contains $N \times L$ learnable parameters in the differentiable wavetable D during training. During inference, the waveforms in D can be regarded as static waveforms just like the sinusoidal functions for the additive synthesizer in DDSP. Thus our model has a comparable number of parameters to DDSP [45] at inference time. We set L to 512 which we find is enough for reconstructing audios with a 16kHz sampling rate. All the parameters in our model are trained jointly by minimizing the reconstruction error of the autoencoder.

3.5.8 Loss

We use a multi-scale spectral loss similar to [45]:

$$L_{\text{reconstruction}} = \sum_i ||S_i - \hat{S}_i||_1, \tag{3.9}$$

where S_i and $\hat{S}_i$ respectively denote the (magnitude) spectrograms of the target audio and the synthesized audio, and $i \in \{2048, 1024, 512, 256, 128, 64\}$ represents the FFT size used to calculate the spectrograms. Note that unlike [45], we exclude the log term $|| \log S_i - \log \hat{S}_i||_1$ as it causes training instability. We find this modification does not influence the quality of the synthesized audio.

3.6 Related Work

3.6.1 Wavetable Synthesis (WTS)

Wavetable synthesis generates audio from a collection of static, single-period waveforms called "wavetables", that each capture a unique harmonic spectrum. Wavetables are typically 256 - 4096 samples in length; a collection can contain a few to several hundred wavetables depending on the use case. Periodic waveforms are synthesized by indexing into the wavetables as a lookup table and interpolating between neighboring samples. WTS was historically used on early hardware synthesizers where memory and computing power were limited. Today, WTS continues to underpin commercial sound design and synthesis tools due to its ability to generate a wide variety of timbres coupled with efficient performance. Wavetables are normally hand-crafted or extracted programmatically from audio spectra [15, 77]. In this work, we learn data-driven wavetables.

3.6.2 Differentiable dictionaries

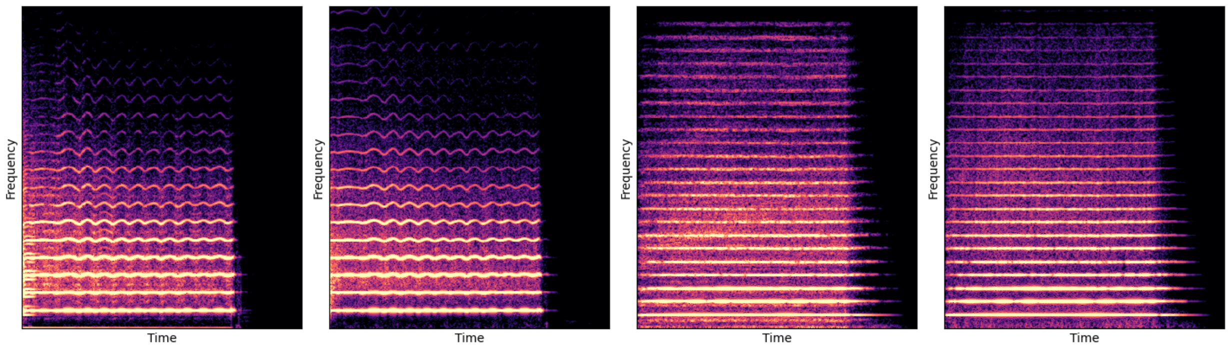
ML models can be equipped with differentiable memory or dictionaries that are learned end-to-end with model parameters. Models can write to external memory to record and lookup changes in state [65]. Discriminative models can learn a differentiable dictionary for improved classification during inference [170, 213, 212]. Here, we formulate wavetables as a differentiable dictionary optimized jointly with model parameters.

3.6.3 Differentiable Digital Signal Processing

DDSP [45] describes a family of techniques utilizing strong inductive biases from DSP combined with modern ML. Successful applications include audio synthesis [45], pitch detection [46], artificial reverberations [104], IIR filters [13] and audio effect manipulation [157]. Building on these works, we add WTS as a new technique for generative audio tasks.

Table 3.1: Reconstruction error comparison on the Nsynth dataset. Compared to the SOTA additive synthesis approach [45], our wavetable synthesis approach achieves comparable or lower errors.

DDSP Additive Synthesis	Wavetable Synthesis (Ours) with N=			
	5	10	20	100
0.5834 ± 0.0035	0.6448 ± 0.0041	0.5989 ± 0.0042	$\mathbf{0.5712} \pm \mathbf{0.0037}$	0.5756 ± 0.0034



(a) Target (b) Reconstructed (c) Target (d) Reconstructed

Figure 3.3: Spectrograms of two target samples and their corresponding reconstruction.

3.7 Experiments

We benchmark against the original DDSP autoencoder [45], where a DNN controls an additive synth and filtered noise synth to produce harmonic and non-harmonic components of audio respectively. We replace the additive synth with our wavetable synth and use an identical filtered noise synth. Noise generation is a stochastic process that must be modeled separately. Like [45], we omit the optional reverb module when training on the NSynth dataset [47].

Dataset We use the same subset of the NSynth dataset [47] in [45, 69, 48], containing 70,000 mono 16kHz samples each 4 seconds long. The examples comprise mostly strings, brass, woodwinds, and mallets. At 16kHz, a wavetable length $L = 512$ is enough to represent all harmonics at the lowest fundamental frequency of interest (20Hz).

Table 3.1 reports the reconstruction error of the SOTA additive-based autoencoder from [45] and the proposed DWTS-based autoencoder. We vary the number of wavetables N =

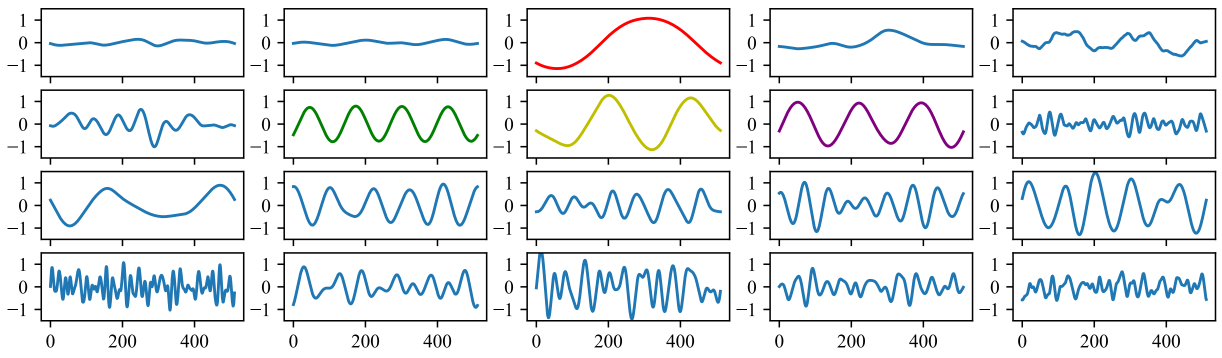


Figure 3.4: Learned wavetables ordered with highest average attention weights appearing first (normal English reading order). Wavetables of key harmonics are highlighted: f_0 (red), f_1 (yellow), f_2 (purple) and f_3 (green). The remaining wavetables are data-driven combinations of higher harmonics. The first two wavetables appear to be silent.

$5, 10, 20, 100$. Our approach achieves the lowest reconstruction error of 0.5712 using only 20 wavetables. At the expense of a small reduction in quality compared to the baseline, N can be low as 10. Fig 3.3 shows spectrograms of two samples and their matching reconstructions using wavetables. We encourage readers to listen to the online supplement[1].

Crucially, the wavetables in D form an alternative, compact set of basis vectors spanning an L-dimensional space extracted directly from the data. When N is very small at 5, reconstruction suffers due to an insufficient number of bases. 10-20 wavetables strike an optimal balance for the NSynth dataset. Wavetables reduce the number of control dimensions by an order of magnitude compared to the 100 sinusoids in an additive synth [45]. More importantly, the extracted wavetables belong to an explicit dictionary that is portable to other tasks. We show this property in later sections. The idea of basis decomposition is also successfully applied to other tasks such as convolution kernel decomposition [205, 207].

3.7.1 Visualizing Wavetables

Fig 3.4 shows learned wavetables from the NSynth dataset when $N = 20$. Despite being initialized with noise, the learned wavetables are smooth and diverse in shape. They also match the physics of NSynth sounds. In acoustic instruments, energy is focused on lower frequencies,

[1]https://lamtharnhantrakul.github.io/diffwts.github.io/

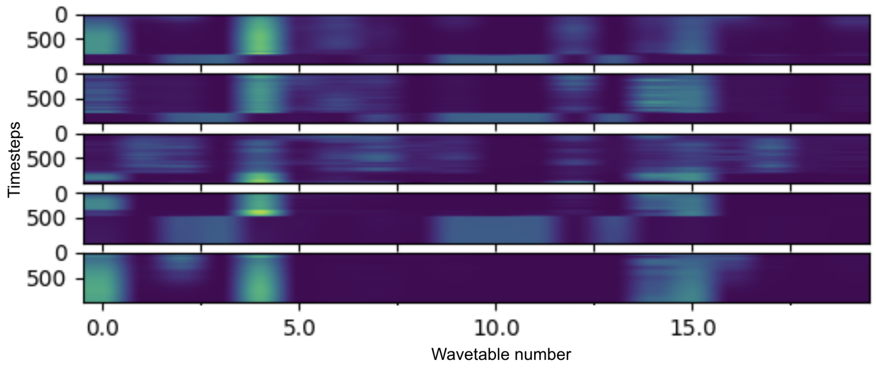
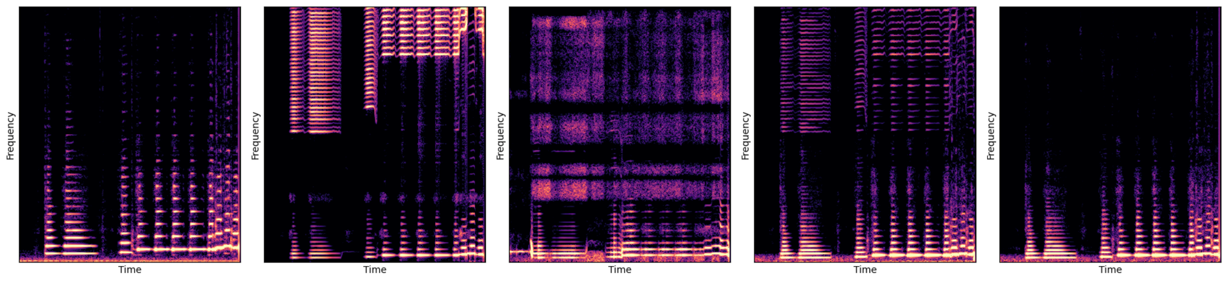


Figure 3.5: Visualization of the time-varying attention weights of 5 samples using 20 wavetables.



(a) Original (b) **Add Scratch** (c) **Add Pretrain** (d) **DWTS Scratch** (e) **DWTS Pretrain**

Figure 3.6: Spectrograms of original audio (a) and synthesized samples from an input $f_0(n)$ pitch shifted down by an octave (b-e)

particularly the first few harmonics, compared to higher harmonics [160]. Wavetables in Fig 3.4 are ordered with the highest average attention weights appearing first. The wavetable highlighted in red is a phase-shifted sinusoid of one period i.e. the fundamental frequency f_0. Other key partials f_1, f_2 and f_3 are highlighted in yellow, purple and green. The remaining wavetables are data-driven combinations of higher harmonics, compactly summarizing in a single wavetable entry what would have taken several sinusoidal components in an additive synth [45] to represent. Fig 3.5 shows the attention weights over time for five audio samples. The attention visibly shifts across wavetables to output the desired spectrum.

The asymmetric wavetables are the result of complex behavior in magnitude and phase. We found phase-locking wavetables to start and end at 0 deteriorated performance. It suggests the model takes advantage of phase relationships within and between wavetables. This precise control of wavetable phase will be particularly valuable in future work exploring the synthesis of stereo and binaural audio [159].

3.7.2 One shot learning and audio manipulations

In domains such as Neural Machine Translation (NMT), word embeddings [138] extracted from end-to-end language learning are useful for other language tasks. Analogously, we reason our data-driven wavetables should be useful in other scenarios like one-shot learning and data-efficient extrapolation. Unlike an implicit multi-dimensional vector, wavetables are an explicit and interpretable representation.

One-shot setup: Given only a single 4-second passage of saxophone from the URMP dataset [105], we train a new autoencoder model initialized with pretrained wavetables from Fig 3.4 (*DWTS Pretrain*). This model only outputs time-varying attention weights, since the wavetables are now a fixed dictionary lookup. We compare against three baselines: (1) additive-synth autoencoder trained from scratch (*Add Scratch*), (2) finetuning an additive-synth autoencoder pretrained on Nsynth (*Add Pretrain*) and (3) Wavetable-synth autoencoder trained from scratch (*DWTS Scratch*).

Pitch extrapolation: While all models achieve identical high-quality one-shot reconstructions of the saxophone segment, only *DWTS Pretrain* is robust to overfitting during extrapolation. Fig 3.6 shows how all baselines exhibit high-frequency artifacts when input $f_0(n)$ is shifted. *DWTS+Pretrain* remains artifact free even at extreme shifts, such as 3 octaves below the original sample.

We repeat with a 4-second piano passage. A piano is challenging to model due to the presence of both many harmonics and percussive hammer hits [160]. We also compare against the Librosa library pitch shift function based on traditional DSP [14]. When resynthesizing the segment 3 octaves down, *DWTS Pretrain* is the only method that preserves the hammer's percussive impact and independently shifts harmonic components. Librosa pitch shift loses the transient impact completely.

Connection to PSOLA: In this scenario, we hypothesize *DWTS Pretrain* approaches an optimal Pitch Synchronous Overlap and Add (PSOLA) algorithm [26]. PSOLA windows a single cycle of the original waveform in the time domain, re-patching and overlapping these windows at the new pitch. Imperfections in this windowing and overlapping process can cause artifacts. *DWTS Pretrain* guarantees single-cycle waveforms in D, where re-pitching is trivially achieved using a slower phase accumulator $\tilde{\phi}(n)$ reading through a wavetable $_i$. This opens up applications like data-efficient neural sampling, pitch correction and efficient polyphony using multiple phase accumulators. We leave wavetables extracted from speech or singing voices for future work.

3.7.3 Computational Complexity

Realtime performance is determined by many factors including algorithmic complexity, low-level optimizations, target hardware and the C++ ML framework used for matrix operations. Here, we consider only the key difference between the additive synth-based pipeline in [45], which already runs real-time [60], and our new approach.

A.) *Additive*: a bank of 100 sinusoids where harmonic coefficients are updated at 250 frames per second (FPS) [45]

B.) *DWTS*: 10 pre-learned wavetables where wavetable weights are also updated at 250 FPS.

The remaining elements of each approach are assumed identical in specification and performance. There are two clear areas of improvement with our new method. Firstly, *DWTS* requires only ten interpolated wavetable read operations per sample, compared to the 100 required for *Additive*. Secondly, both pipelines require frame-wise linear smoothing of control data to avoid unwanted artifacts in the synthesized signals. *Additive* requires all 100 sinusoidal amplitude coefficients to be smoothed per sample, whereas *DWTS* requires smoothing on only 10. We confirm this on a 2.6GHz 2019 Macbook Pro. Over 10k trials, *Additive* takes on average 251.32ms to generate all 250 frames in 1 second of audio, whereas *DWTS* takes only 20.19ms. Therefore, DWTS has significantly lower computational complexity, which is important to deploy our machine learning model on edge devices [103, 227, 130, 131, 82, 132, 224, 211, 84, 85], such as smartphones, tablets, laptops, sensors, gateways, and Internet of Things (IoT) device.

3.8 Closing Remarks

In this chapter, we presented a differentiable wavetable synthesizer capable of high-fidelity neural audio synthesis. Similar to the previous chapter where we learn a dictionary of related neighbors to fine-tune the model and improve the prediction performance, here we learn a dictionary of related waveform instances and train a model to determine which instances are related for synthesis at different times. We demonstrate how explicit and learnable wavetables offer many advantages including robust one-shot learning and an order-of-magnitude reduction in computational complexity. The approach opens up applications such as data-efficient neural audio sampling and pitch-shifting.

Regarding limitations, as our model DWTS is based on Wavetable Synthesis, DWTS can only synthesize harmonic sounds. In contrast, DDSP is based on Additive Synthesis and thus it can also synthesize in-harmonic sounds. Another limitation of DWTS is that it is an auto-encoder model, which means it is not a generative model that is capable of synthesizing unseen audio timbres. Thus, future works can combine DWTS with a generative model to synthesize novel audio timbres.

CHAPTER 4: LEVERAGING RELATED POINTS IN A COLLECTION OF CELLS FOR CLASSIFICATION

4.1 Notation

- $x^{(i)} \in \mathbb{R}^d$: the vector of d features (e.g. proteins or genes) measured in cell i.
- $\mathcal{X} = \{x^{(i)}\}_{i=1}^{n}$: a sample-set, which is a collection of n cells from an individual's blood or tissue.
- $\mathcal{D} = \left\{\mathcal{X}^{(k)}\right\}_{k=1}^{N} = \left\{\{x^{(k,i)}\}_{i=1}^{n_k}\right\}_{k=1}^{N}$: a multi-sample dataset $\mathcal{D}$ contains multiple sample-sets (across multiple, N, individuals and conditions). $\mathcal{X}^{(k)} = \{x^{(k,i)}\}_{i=1}^{n_k}$ is the sample-set for the k-th profiled biological sample.
- $\mathcal{D} = \left\{(\mathcal{X}^{(k)}, y^{(k)})\right\}_{k=1}^{N}$: dataset consists of sample-set and label tuples. $y^{(k)}$ is the label.

4.2 Introduction

In this chapter, we discuss the case where the input data is a collection of instances rather than a single instance. In such a case, making a prediction is more challenging as we need to compare across multiple sets of multiple vectors and consider the related instances in a set together to infer the label of the set.

Machine learning methods have been used in biomedical data analysis where the input data is a single instance such as a medical image [225, 50, 201, 52, 116, 118, 54, 99, 98, 119]. Here, we consider a collection of instances and begin with the single-cell sample-set classification problem. Modern immune profiling techniques such as flow and mass cytometry (CyTOF) enable comprehensive profiling of immunological heterogeneity across a multi-patient cohort [38, 117]. In

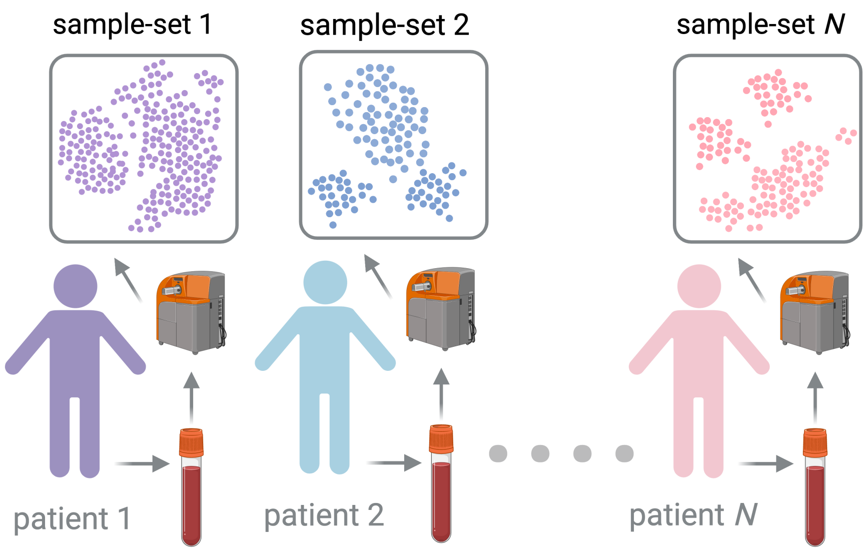


Figure 4.1: Single-cell profiles produce a *sample-set* of (typically thousands of) multidimensional feature vectors of measured features per cell (illustrated as scatter plots). Thus, a dataset of multiple biological samples will result in a dataset of multiple sets (shown above).

recent years, such technologies have been applied to numerous clinical applications. In particular, these assays allow for both the phenotypic and functional characterization of immune cells based on the simultaneous measurement of 10-45 protein markers [11]. Modern single-cell flow and mass cytometry technologies measure the expression of several proteins of the individual cells within a blood or tissue sample. Each profiled biological sample is thus represented by a set of hundreds of thousands of multidimensional cell feature vectors, which incurs a high computational cost to predict each biological sample's associated phenotype with machine learning models. Such a large set cardinality also limits the interpretability of machine learning models due to the difficulty in tracking how each individual cell influences the ultimate prediction.

To comprehensively profile the immune systems of biological samples collected across multiple individuals, a unique data structure of multiple *sample-sets* is ultimately produced. Each sample-set is likely to contain hundreds of thousands of cells collected from an individual, and machine learning models are often used to predict their associated clinical or experimental labels. This concept is illustrated in Fig. 4.1. In profiling multiple individuals with a single-cell technology, a *dataset of sets* is produced and is a non-traditional data structure that breaks the standard convention of a *dataset of feature vectors*. This produces complexity for learning over individuals (e.g. patients), as single-cell data requires a model to characterize and compare across multiple sets of multiple vectors (where each vector represents a cell). Another challenge of analyzing single-cell data is the large number of cells in each sample-set, which incurs a computational cost and obscures the decision-making process of machine learning models.

To efficiently and accurately link cellular heterogeneity to clinical outcomes or external variables, here we introduce Cell Kernel Mean Embedding (CKME), a method based on kernel mean embeddings [140] that directly featurizes cells in samples according to their aggregate distribution. For classification tasks, we train a linear classifier (e.g. linear SVM) in the KME feature space. Despite its simplicity, CKME achieves comparable to state-of-the-art gating-free performance on three flow and mass cytometry datasets with multiple sample-sets and associated clinical outcomes. We show that CKME may be used to obtain an individual cell score for each cell in a sample-set, where the ultimate prediction is the result of the average cell score. As a result, CKME is highly interpretable since one can quantify the individual contribution of every cell to the final prediction. Predictions are then synthesized further using kernel herding [32], which identifies key cells to retain in a subsample. We leverage the transparency of CKME in a thorough analysis to understand predictions. First, we study the semantics of cell scores from CKME, and show that they follow several desirable, intuitive properties. After, we explore cell scores to obtain biological insights from our models, and show that discoveries from CKME are consistent with existing knowledge.

4.3 Problem Formulation

Given a dataset of multiple sample-sets as discussed above, we wish to build a discriminative model to predict patient-level phenotypes or conditions based on the cellular composition that is found within a sample-set, $\mathcal{X} \to y$. If the model is human interpretable, it will help researchers make sense of how the cell feature vectors in a sample-set influence the predicted phenotype or conditions of a patient, which eventually will lead to an improved understanding of biological phenomena and enable better diagnoses and treatment for patients. Below we propose a methodology to featurize and classify input sample-sets in a way that is more transparent and human-understandable than comparably performant models (e.g. [221]).

4.4 Background

Our method is built on three techniques, namely Kernel Mean Embedding for featurization of sample-sets, Random Fourier Feature for computational cost reduction and Kernel Herding for selecting representative instances.

4.4.1 Kernel Mean Embedding

Kernel methods have achieved great success in many distinct machine learning tasks, including: classification [35], regression [196], and dimensionality reduction [137]. They utilize a positive definite kernel function $\mathtt{k} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ [1], which induces a reproducing kernel Hilbert space (RKHS) (e.g. see [12] for further details). Kernels have also been deployed for representing a distribution, p, with the *kernel mean embedding* $\mu_p : \mathbb{R}^d \mapsto \mathbb{R}$:

$$\mu_p(\cdot) \equiv \mathbb{E}_{x \sim p}[\mathtt{k}(x, \cdot)]. \tag{4.1}$$

Note that μ_p is itself a function (see e.g. [140] for more details). For "*characteristic*" kernels, $\mathtt{k}$, such as the common radial-basis function (RBF) kernel $\mathtt{k}(x, x') = \exp(-\frac{1}{2\gamma}||x - x'||^2)$, the kernel mean embedding will be unique to its distribution; i.e., for characteristic kernels, $||\mu_p - \mu_q|| = 0$ if and only if $p = q$. In general, the distance[2] $||\mu_p - \mu_q||$ induces a divergence, the maximum mean discrepancy (MMD) [67], between distributions.

[1]Note that kernels may be defined over non-real domains, this is omitted for simplicity.

[2]In the RKHS norm.

4.4.2 Random Fourier Features

To get a non-linear decision boundary, we can map data into a high-dimensional space such that the data is separable in that space. However, explicitly calculating that mapping is not scalable. To solve this problem, the kernel trick uses kernel functions $\mathtt{k}(x, y)$ to calculate the dot product in the high-dimensional space without explicitly mapping data to that space: $\mathtt{k}(x, y) = \langle \phi(x), \phi(y) \rangle$. However, this requires to perform kernel evaluation on all pairs of datapoints, which scales quadratically to dataset size. Instead of relying on the implicit lifting provided by the kernel trick, the Random Fourier Features (RFF) approach [153] explicitly maps the data to a Euclidean inner product space using a randomized feature map z so that the inner product between a pair of transformed points approximates their kernel evaluation: $\mathtt{k}(x, y) = \langle \phi(x), \phi(y) \rangle \approx z(x)'z(y)$.

4.4.3 Kernel Herding

Let $x \in \mathcal{X}$ denotes some state over an index set $\mathcal{X}$ and let $\phi : \mathcal{X} \to \mathcal{H}$ denote a feature map into a Hilbert Space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle$. Given a probability distribution $p(x)$, herding [31] consists of the following processes to sequentially generate pseudo-samples x:

$$x_{t+1} = \underset{x \in \mathcal{X}}{\operatorname{argmax}} \langle w_t, \phi(x) \rangle \tag{4.2}$$

$$w_{t+1} = w_t + \mathbb{E}_{x \sim p}[\phi(x)] - \phi(x_{t+1}) \tag{4.3}$$

Give suitable assumptions and the further restrictions of finite-dimensional discrete state spaces [214], we can show that herding greedily minimizes the following error

$$||\mathbb{E}_{x \sim p}[\phi(x)] - \frac{1}{T} \sum_{t=1}^{T} \phi(x_t))||^2. \tag{4.4}$$

Thus, herding generate pseudo-samples that best approximate $\mathbb{E}_{x \sim p}[\phi(x)]$.

The herding algorithm is extended to continuous spaces by using the kernel trick and the resulting method is called kernel herding [32]. This method is an infinite memory deterministic process that learns to approximate a PDF with a collections of samples. It is shown that kernel herding decreases the error at a rate $O(\frac{1}{T})$ which is much faster than the usual $O(\frac{1}{\sqrt{T}})$.

4.5 Methods

To classify labels of interest, y, given an input sample-set, $\mathcal{X}$, we featurize $\mathcal{X}$ so that we may learn an estimator over those features. However, unlike traditional data-analysis, which featurizes a single vector instance $x \in \mathbb{R}^d$ with features $\phi(x) : \mathbb{R}^d \mapsto \mathbb{R}^q$, here we featurize a *set of multiple* vectors (one vector for each cell in a sample-set) $\mathcal{X} = \{x^{(i)}\}_{i=1}^n$, $\phi(\mathcal{X}) \in \mathbb{R}^q$. Featurizing a set presents a myriad of challenges since typical machine learning approaches are constructed for *statically-sized, ordered* inputs. In contrast, sets are of *varying cardinalities* and are *unordered*. Hence, straight-forward approaches, such as concatenating the sample-set elements into a single vector $(x^{(1)}, \ldots, x^{(n)}) \in \mathbb{R}^{nd}$ shall fail to provide mappings that do not depend on the order that elements appear in. To respect the *unordered* property of sample-sets one must carefully featurize $\mathcal{X}$ in a way that is *permutation-invariant*. That is, the features $\phi(\mathcal{X})$ should be unchanged regardless of what order that the elements of $\mathcal{X}$ are processed. Recently, there have been multiple efforts to create methods based on neural networks to featurize sets in a permutation invariant manner [150, 223, 174, 114, 171]. Although these approaches provide expressive, non-linear, discriminative features, they are often opaque and difficult to understand in how they lead to their ultimate predictions. In contrast, we propose an approach based on kernels and random features that is more transparent and understandable whilst being comparably accurate.

For our purposes, we propose to use kernel mean embeddings to featurize sample-sets:

$$\mu_{\mathcal{X}}(\cdot) \equiv \frac{1}{n}\sum_{i=1}^{n} \mathrm{k}(x^{(i)}, \cdot) \approx \mu_p(\cdot), \tag{4.5}$$

where p is the underlying distribution (of cell features) that $\mathcal{X}$ was sampled from. That is, the set embedding $\mu_{\mathcal{X}}$ (eq. 4.5) also approximately embeds the underlying distribution of cells that the sample-set was derived from. To produce a real-valued output from the mean embedding, one would take the (RKHS) inner product with a learned function f:

$$\langle \mu_{\mathcal{X}}, f \rangle = \frac{1}{n} \sum_{i=1}^{n} \langle \mathrm{k}(x^{(i)}, \cdot), f(\cdot) \rangle = \frac{1}{n} \sum_{i=1}^{n} f(x^{(i)}), \tag{4.6}$$

where the last term follows from the reproducing property of the RKHS. For example, eq. 4.6 can be used to output the log-odds for a target y given $\mathcal{X}$: $p(y = 1|\mathcal{X}) = (1 + \exp(-\langle \mu_{\mathcal{X}}, f \rangle))^{-1}$. Using the representer theorem [167] it can be shown that f may be learned and represented using a "*Gram*" matrix of pairwise kernel evaluations, $\mathrm{k}(x, x')$. This, however, will be prohibitive in larger datasets. Instead of working directly with a kernel k, we propose to leverage random Fourier features for computational efficiency and simplicity.

We propose to use random Fourier frequency features [153] to build our mean embedding [140]. For a shift-invariant kernel (such as the RBF kernel), random Fourier features provide a feature map $\varphi(x) \in \mathbb{R}^D$ such that the dot product in feature space approximates the kernel evaluation, $\varphi(x)^T \varphi(x') \approx \mathrm{k}(x, x')$. I.e. $\varphi(x)$ acts as an approximate *primal space* for the kernel k (please see [153, 184, 185, 143, 144] for further details). Using the dot product of $\varphi(x)$, our mean embedding becomes:

$$\mu_{\mathcal{X}} = \frac{1}{n} \sum_{i=1}^{n} \varphi(x^{(i)}) \in \mathbb{R}^D, \quad \mu_{\mathcal{X}}(x') = \frac{1}{n} \sum_{i=1}^{n} \varphi(x^{(i)})^T \varphi(x') \tag{4.7}$$

where $\varphi(x) = \Big(\sin(\omega_1^T x), \ldots, \sin(\omega_{D/2}^T x), \cos(\omega_1^T x), \ldots, \cos(\omega_{D/2}^T x)\Big)$ with random frequencies $\omega_j \sim \rho$ drawn once (and subsequently held fixed) from a distribution ρ that depends on the kernel k. For instance, for the RBF kernel, ρ is an *iid* multivariate independent normal with mean 0 and variance that depends (inversely) on the bandwidth of the kernel, γ.

When computing the mean embedding in the $\varphi(\cdot)$ feature space (eq. 4.7), one may directly map $\mu_{\mathcal{X}}$ to a real value with a dot product with learned coefficient $\beta \in \mathbb{R}^D$:

$$\mu_{\mathcal{X}}^T \beta = \left(\frac{1}{n} \sum_{i=1}^{n} \varphi(x^{(i)}) \right)^T \beta = \frac{1}{n} \sum_{i=1}^{n} \varphi(x^{(i)})^T \beta. \tag{4.8}$$

That is, we may learn a linear model directly operating over the D dimensional feature vectors $\mu_{\mathcal{X}}$, which are composed of the average random features found in a sample-set. Below we expound on how to build a discriminative model based on $\mu_{\mathcal{X}}$.

Linear Classifier with Interpretable Scores With the mean embedding of sample-sets $\{\mu_{\mathcal{X}^{(k)}}\}_{k=1}^N$, $\mu_{\mathcal{X}^{(k)}} \in \mathbb{R}^D$, we can build a discriminative model $f : \mathbb{R}^D \to \mathbb{R}$ to predict their labels. If we choose f as a linear model (e.g. linear SVM), then $f(\mu_{\mathcal{X}})$ can be generally expressed as

$$f(\mu_{\mathcal{X}^{(k)}}) = \mu_{\mathcal{X}^{(k)}}^T \beta + b = \frac{1}{n_k} \sum_{i=1}^{n_k} \underbrace{\varphi(x^{(k,i)})^T \beta + b}_{s^{(k,i)}} = \frac{1}{n_k} \sum_{i=1}^{n_k} s^{(k,i)}, \tag{4.9}$$

where $\beta \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are the weight and the bias of the linear model (see Fig.4.2 for illustration). From eq. 4.9, we can express the output response of f as the mean of all $s^{(k,i)}$, which we denote as *the score* of the i-th cell in the k-th sample-set $\mathcal{X}^{(k)}$. This formulation naturally allows to quantify and interpret the contribution of every cell to the final prediction, which can potentially lead to an improved understanding of biological phenomena and enable better diagnoses and treatment for patients.

Kernel Herding When using random features (eq. 4.7), $\mu_{\mathcal{X}}$ may be understood as the average of random features for cells found in a sample-set. Predicted outputs based on $\mu_{\mathcal{X}}$ (eq. 4.9) may be further interpreted as the average "score" of cells, $s^{(i)}$, in the respective sample-set. However, sample-sets may contain many (hundreds of) thousands of cells, making it cumbersome to analyze and synthesize the cell scores in a sample-set. To ease interpretability, we propose to subselect cells in the sample-set in a way that yields a similar embedding to the original sample-set. That is, we wish to find a subset $\hat{\mathcal{X}} \subset \mathcal{X}$ such that $\mu_{\hat{\mathcal{X}}} \approx \mu_{\mathcal{X}}$, which implies that one may make

Algorithm 2 COMPUTE THE MEAN EMBEDDING AFTER SUB-SELECTION VIA KERNEL HERDING

```
Require: A sample-set $\mathcal{X}$, number of cells kept after sub-selection $m$, dimensionality of the random feature space $D$, kernel hyperparameter $\gamma$.
1: # Compute Random Fourier Frequency Features
2: Compute $\mathbf{W} \in \mathbb{R}^{d \times \frac{D}{2}}$ by sampling its elements independently $\mathbf{w}_{i,j} \sim \mathcal{N}(0, \frac{1}{\gamma})$
3: for each $x^{(i)} \in \mathcal{X}$ do
4:     $\varphi(x^{(i)}) \leftarrow [\sin(\mathbf{W}^T x^{(i)}), \cos(\mathbf{W}^T x^{(i)})] \in \mathbb{R}^D$, where $[\cdot, \cdot]$ denotes concatenation.
5: end for
6: # Sub-selection with Kernel Herding
7: Initialize $j \leftarrow 1$, $\hat{\mathcal{X}} \leftarrow \emptyset$, $\theta_0 \leftarrow \frac{1}{n}\sum_{i=1}^{n} \varphi(x^{(i)})$, $\theta \leftarrow \theta_0$
8: while $j \leq m$ do
9:     $i^* \leftarrow \arg\max_i \theta^T \varphi(x^{(i)})$
10:    $\hat{\mathcal{X}} \leftarrow \hat{\mathcal{X}} \cup \{\varphi(x^{(i^*)})\}$,
11:    $\theta \leftarrow \theta + \theta_0 - \varphi(x^{(i^*)})$
12:    $j \leftarrow j + 1$
13: end while
14: # Compute the mean embedding of the Sub-selected Sample-set
15: $\mu_{\hat{\mathcal{X}}} = \frac{1}{m}\sum_{z \in \hat{\mathcal{X}}} z$
16: return $\mu_{\hat{\mathcal{X}}}$
```

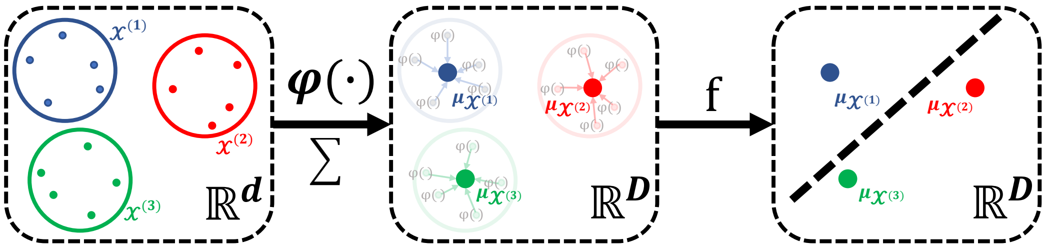


Figure 4.2: The pipeline of CKME to process the dataset and train the classifier. The left rectangle shows a dataset with 3 sample-sets and each sample-set contains 5 features vectors in the original feature space $\mathbb{R}^d$. In the middle rectangle, we use $\varphi(\cdot)$ to transform the data into random Fourier feature space $\mathbb{R}^D$ and compute the mean embedding $\mu_{\mathcal{X}^{(1)}}, \mu_{\mathcal{X}^{(2)}}, \mu_{\mathcal{X}^{(3)}}$ for every sample-set (4.7). Finally, in the right rectangle, we train a linear discriminative model f (4.9) upon the mean embeddings to predict the label of each sample-set.

similar inferences using a smaller (easier to interpret) subset of cells as with the original sample-set. Although a uniformly random subsample of $\mathcal{X}$ would provide a decent approximation $\mu_{\hat{\mathcal{X}}}$ for large enough cardinality ($|\hat{\mathcal{X}}|$), it is actually a suboptimal way of constructing an approximating subset [32]. Instead, we propose to better construct synthesized subsets (especially for small cardinalities) using kernel herding (KH) [32]. KH can provide a subset of m points that approximates the mean embedding as well as m^2 uniformly sub-sampled points. We expound on KH for producing subsets of key predictive cells in Algorithm 3. We denote the dataset with the sub-selected sets as $\hat{\mathcal{D}} = \{(\hat{\mathcal{X}}^{(k)}, y^{(k)})\}_{k=1}^{N}$.

4.6 Related Work

The first class of methods for identifying coherent cell-populations and specifying their associated immunological features are *gating-based*[3], and operate by first assigning cells to populations either manually, or in an automated manner by applying an unsupervised clustering approach [117, 3]. After an individual's cells have been assigned to their respective cell-populations, corresponding immunological features are specified by 1) computing frequencies or the proportion of cells assigned to each population and 2) computing functional readouts as the median expression of a particular functional marker [181]. Though gating-based approaches can be automatic without human intervention, they have several drawbacks: 1) they tend to be sensitive to variation in the parameters of the underlying clustering algorithms [181]. 2) the underlying clustering algorithms typically do not scale well to millions of cells. 3) they typically need to rerun the expensive clustering process whenever a new sample-set is given [93]. Furthermore, partitioning cells into discrete clusters can be inadequate in the context of continuous phenotypes such as drug treatment response and ultimately limits the capacity of having a single-cell resolution of the data.

To address these limitations, the second class of methods are *gating-free* and rely on representing, or making predictions based on individual cells [5, 79, 221]. For example, CytoDx [79] uses a linear model to first compute the response of every single cell individually, then the responses are aggregated by a mean pooling function. Finally, the aggregated result is used by another linear model to predict clinical outcomes. CellCNN [5] uses a 1-d convolution layer to learn filters that are responsive to marker profiles of cells and aggregates the responses of cells to

[3]Gating refers to partitioning cells into populations

different filters via a pooling layer for prediction. Since both CytoDx and CellCNN only contain a single linear or convolution layer before the pooling operation, they enable human interpretation. However, this property also limits their expressive power. To enhance model flexibility, CytoSet [221] employs a deep learning model similar to Deep Set [223] to handle set data with a permutation invariant architecture that stacks multiple intermediate permutation *equivariant* neural network layers. As a result, the sample-set featurization that CytoSet achieves, while discriminative and accurate, is opaque and a black box, making it difficult to analyze downstream for biological discoveries. In addition, CytoSet usually contains millions of parameters, further obfuscating the underlying predictive mechanisms. Our method CKME belongs to this *gating-free* class; we show that CKME achieves interpretability without harming model expressiveness.

4.7 Experiments

4.7.1 Datasets

We used publicly available, multi-sample-set flow and mass cytometry (CyTOF) datasets. Each sample-set consists of several protein markers measured across individual cells.

Preeclampsia. The preeclampsia CyTOF dataset profiles 11 women with preeclampsia and 12 healthy women throughout their pregnancies. Sample-sets corresponding to profiled samples from women are publicly available[4]. Our experiments focused on uncovering differences between healthy and preeclamptic women.

HVTN. The HVTN (HIV Vaccine Trials Network) is a Flow Cytometry dataset that profiled T-cells across 96 samples that were each subjected to stimulation with either Gag or Env proteins [5] [3]. The data are publicly available[6]. Our experiments focused on uncovering differences between Gag and Env stimulated samples.

[4]`http://flowrepository.org/id/FR-FCM-ZYRQ`

[5]Note these are proteins meant to illicit functional responses in immune cell-types.

[6]`http://flowrepository.org/id/FR-FCM-ZZZV`

Table 4.1: Classification accuracies on the three datasets. Standard deviations are computed from 5 independent runs.

	HVTN	Preeclampsia	COVID-19
CytoDx [79]	65.24 ± 1.90	56.17 ± 3.95	68.82 ± 3.15
CellCNN [5]	81.87 ± 1.77	58.51 ± 3.25	75.93 ± 2.62
CytoSet [221]	$\mathbf{90.46 \pm 2.20}$	62.85 ± 3.42	$\mathbf{86.55 \pm 1.76}$
CKME	$\mathbf{90.68 \pm 1.69}$	$\mathbf{63.52 \pm 2.22}$	$\mathbf{86.38 \pm 1.92}$

Table 4.2: Number of parameters ($\times 10^3$) for different methods on different datasets.

	HVTN	Preeclampsia	COVID-19
CytoDx [79]	**0.01**	**0.03**	**0.03**
CellCNN [5]	40.7	20.2	40.5
CytoSet [221]	330.2	80.0	330.0
CKME	2.0	2.0	2.0

COVID-19. The COVID-19 dataset analyzes cytokine production by PBMCs derived from COVID-19 patients. The dataset profiles healthy patients, as well patients with moderate and severe covid cases. Specifically, the dataset consists of samples from 49 total individuals and is comprised of 6 healthy, 23 labeled intensive-care-unit (ICU) with moderately severe COVID, and 20 Ward (non-ICU, but covid-severe) labeled individuals with severe COVID cases, respectively. The data are publicly available[7]. Our experiments focused on uncovering differences between sample-sets from ICU and Ward patients.

4.7.2 Baselines

We focus our comparison to current state-of-the-art gating-free methods CytoDx [79], CellCNN [5], CytoSet [221], which look to assuage the drawbacks of gating-based methods. Please refer to Sec. 4.6 "Related Work" for a detailed discussion about these methods.

[7]`http://flowrepository.org/id/FR-FCM-Z2KP`

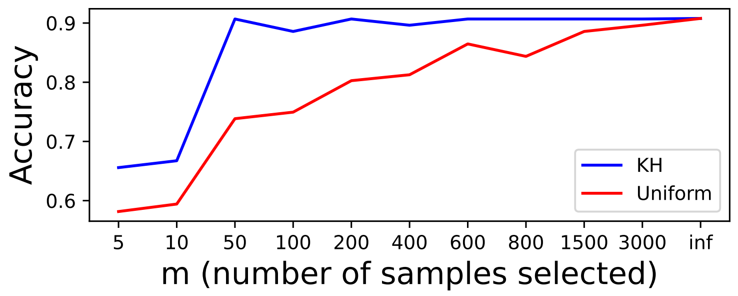


Figure 4.3: The influence of the number of cells selected by KH and uniform sub-sampling, m, on the classification accuracy. $m =$"inf" means no sub-selection.

4.7.3 Implementation Details

Given the limited number of sample-sets, we used 5-fold cross-validation to report the classification accuracies on the three datasets. We tuned hyperparameters (e.g. the bandwidth parameter γ) on the validation set. On the HVTN and the Preeclampsia datasets, γ was set to 1 while on the COVID-19 dataset γ was set to 8. The dimensionality D was set to 2000 for all datasets. The linear classifier in eq. 4.9 is implemented by a linear SVM. For all the methods, $m = 200$ cells are sub-selected for all datasets. The best hyperparameters of CytoSet, CellCNN and CytoDx are selected based on the validation performance. CytoSet, CellCNN, CytoDx was run using their public implementation[8].

4.7.4 Classification Accuracies

We report accuracies in Table 4.1 and numbers of model parameters in Table 4.2. Although our foremost goal is to make a more transparent model, we find that CKME achieves comparable or better accuracies than the state-of-the-art gating-free methods. The strength of CKME's performance is impressive when taking into account that: 1) CKME contains on average two orders of magnitude fewer parameters than CytoSet; 2) unlike CytoSet, which has an opaque, uninterpretable set-featurization and prediction model, CKME uses a simple average score over

[8]CytoSet: `https://github.com/CompCy-lab/cytoset`, CellCNN: `http://eiriniar.github.io/CellCnn/code.html`, CytoDx: `http://bioconductor.org/packages/CytoDx`

Table 4.3: Ablation Studies.

	HVTN	Preeclampsia	COVID-19
CKME w/ Unif	80.26 ± 1.53	55.52 ± 4.33	84.28 ± 1.74
Naive Mean	64.24 ± 3.17	57.60 ± 4.20	83.07 ± 2.29
CKME	$\mathbf{90.68 \pm 1.69}$	$\mathbf{63.52 \pm 2.22}$	$\mathbf{86.38 \pm 1.92}$

cells; 3) comparably simple models, such as CytoDX and CellCNN, have significantly poorer accuracies than CKME. We believe that CKME's success is due to the expressive ability of random Fourier features, which provide flexible non-linear mappings (to cell scores) over input cell features without the need to learn additional featurization (e.g. from a neural network). Moreover, the average of the random Fourier features, the mean map embedding (Sec. "Random Fourier Features"), is descriptive of the overall distribution and composition of respective sample-sets; since the discriminator is trained directly on the mean map embeddings, CKME is able to classify based on the cell composition of sample-sets. Lastly, when a sample-set is sub-selected with kernel herding (Sec. "Kernel Herding"), the selected cells are explicitly a salient, descriptive subset that distills the sample-set whilst preserving overall characteristics.

4.7.5 Ablation Studies

Ablation Studies on Sub-sampling We first ablate the number of sub-samples to study the impact of cell sub-selection on accuracy. Fig. 4.3 shows the classification accuracies of CKME on the HVTN dataset with different numbers of cells selected by Kernel Herding (KH) and uniform sub-selection. We found that KH consistently outperforms uniform sub-selection and the accuracy of KH quickly saturates with as few as 50 cells sub-selected for every sample-set, confirming the strong capacity of Kernel Herding to maintain the distribution of the original sample-sets after sub-selection. Moreover, in Table 4.3, we show the performance of CKME with 200 cells sub-selected by uniform sub-sampling (CKME w/ Unif). Compared to sub-selection with Kernel Herding, we find that using uniform sub-sampling leads to a worse result. This confirms that Kernel Herding is more effective than uniform sub-sampling.

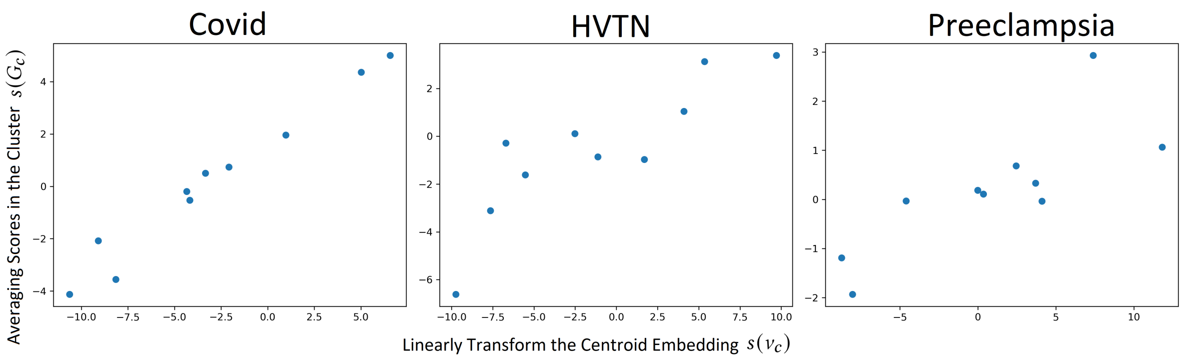


Figure 4.4: Cluster scores computed by the two methods are positively correlated.

Ablation Studies on Naive Mean Embedding Here, we investigate the performance of mean embeddings without using random Fourier features. In this case, we still used Kernel Herding to sub-select m cells but represented the summarized sample-set in the original feature space as $\bar{\mathbf{x}}^{(k)} = \frac{1}{m}\sum_{j=1}^{m}\hat{x}^{(k,j)} \in \mathbb{R}^d$. We call this approach "Naive Mean". As shown in Table 4.3, "Naive Mean" is worse than CKME, which indicates that simple summary statistics ($\bar{\mathbf{x}}^{(k)}$, the mean of input features) does not suffice for classification; in contrast, the random feature kernel mean embedding is able to provide an expressive enough summary of sample-sets.

4.8 Analysis

4.8.1 Semantic Analysis of Cell Scores

Recall that CKME can compute a score, $s^{(k,i)}$ (eq. 4.9), for every cell feature vector $\hat{x}^{(k,i)}$ in the KH sub-selected sample-set $\hat{\mathcal{X}}^{(k)}$. Although this already enables our model to be more transparent, we hope that these scores are semantically meaningful and consistent with intuitive properties that one would want. We draw an analogy to natural language processing word-embeddings [139], where one constructs word-level featurizations (embeddings) that one hopes are semantically meaningful. For example, the sum of a group of consecutive words-embeddings in a sentence should yield a sentence-embedding that maintains the characteristics of that sentence as a whole. Here, we study the semantics of scores of regions (groups) of cells induced by our cell scores.

We may assign scores to regions in two ways: 1) averaging scores of cells in a nearby region ; 2) directly compute the score of the centroid of the respective region. That is, for cells in a nearby region, $G = \{x^{(i)}\}_{i=1}^{m}$, their average score is $s(G) = \frac{1}{m}\sum_{i=1}^{m}\varphi(x^{(i)})^T\beta + b$, where $\varphi : \mathbb{R}^d \mapsto \mathbb{R}^D$ are the Fourier features, and β, b are the parameters of the learned linear model. In contrast, one may *directly* compute the score of the centroid of this region, $\nu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$, as $s(\nu) = \varphi(\nu)^T\beta + b$ (note the abuse of notation on s). Both $s(G)$ and $s(\nu)$ score a region. Analogously to word-embeddings we hope that the score of cells ("words") retains the semantics of the regions ("sentences"). Below we study the semantic retention of cell scores through a comparative analysis between $s(G)$ and $s(\nu)$, and a predictive analysis.

Regions We construct nearby regions with a k-means cluster analysis of cells in each dataset. I.e., we obtain C cluster centroids, $\nu_1, \ldots, \nu_C$, and respective partitions $G_c = \{x \in \bigcup_{k=1}^{N} \hat{\mathcal{X}}^{(k)} \mid c = \text{argmin}_l ||\nu_l - x||\}$. Below we set $C = 10$. The distribution of cluster assignments among cells from positive and negative classes on HVTN is shown in Fig. 4.5. One can see that clusters with a negative score (e.g. Clusters 1 & 3) tend to be more heavily represented in negative sample sets than in positive sample sets (and vice-versa for positive clusters such as Cluster 9).

Comparative Analysis First, we analyze the semantic retention of scores by studying the correlation between the direct scores of cluster centroids $s(\nu_c) = \varphi(\nu_c)^T\beta + b$, to the average score of cells in respective clusters $s(G_c) = \frac{1}{|G_c|}\sum_{x \in G_c}\varphi(x)^T\beta + b$. *Both $s(\nu_c)$ and $s(G_c)$ assign scores to the same region*, however, due to the nonlinearity of Fourier features there is no guarantee that $s(G)$ will match $s(\nu)$ as generally $\frac{1}{m}\sum_{i=1}^{m}\varphi(x^{(i)})^T\beta + b \neq \varphi\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right)^T\beta + b$. Notwithstanding, if cell scores are semantically meaningful, then $s(\nu_c)$ and $s(G_c)$ should relate to each other as they both score regions. When observing the scatter plot of $s(\nu_c)$ vs. $s(G_c)$ across our datasets (see Fig. 4.4), we see that both scores on regions are heavily correlated; this shows that our scores are capable of semantically retaining the characteristics of regions. The respective

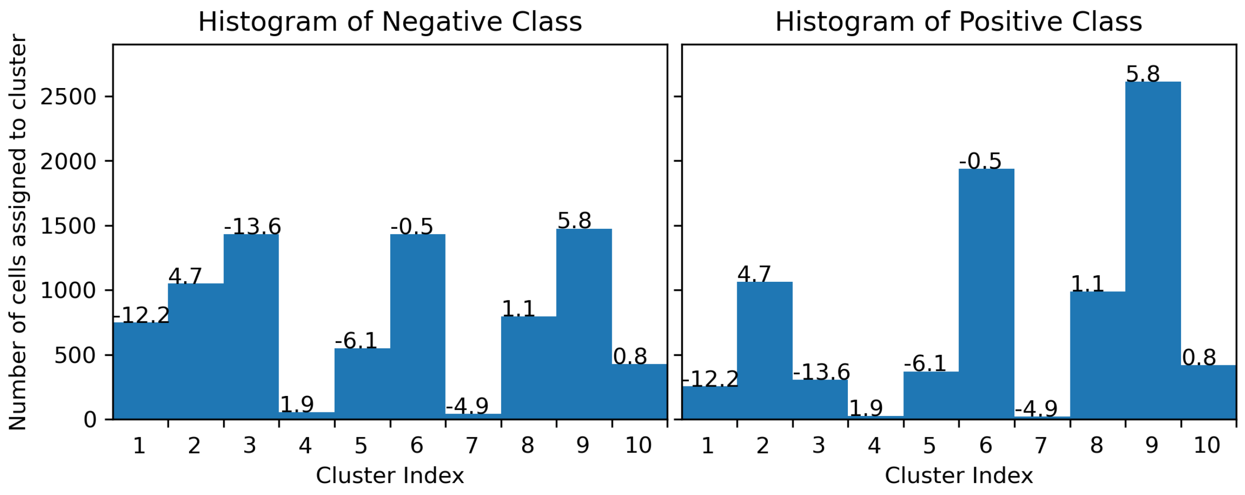


Figure 4.5: The histogram of the clustering assignments on the HVTN dataset. The scores of all clusters computed by linearly transforming the corresponding centrioid embedding, $s(\nu_c)$, is shown on the top of each corresponding bar.

Table 4.4: Classification accuracies on the three datasets. Standard deviations are computed from 5 independent runs.

	HVTN	Preeclampsia	COVID-19
Linear	78.66 ± 2.86	56.64 ± 4.16	76.92 ± 2.05
Centroid CKME	76.48 ± 2.62	60.57 ± 3.19	77.88 ± 2.11
Average CKME	74.51 ± 2.46	60.80 ± 3.03	76.17 ± 2.13
Average MELD	64.52 ± 0.92	59.64 ± 3.47	59.16 ± 2.72

Pearson correlation coefficients for COVID-19, HVTN, and Preeclampsia are 0.9785, 0.8234, and 0.7654.

Predictive Analysis Next, we further study the semantic retention of cell scores with a predictive analysis. Traditionally [17, 151, 181] one may predict a label y using a cluster analysis through a frequency-featurization. E.g. with a linear model $\hat{f}(\hat{\mathcal{X}}^{(k)}) = \sum_{c=1}^{C} \hat{r}_c^{(k)} \alpha_c + a$, where $\hat{r}_c^{(k)}$ is the proportion of cells in $\hat{\mathcal{X}}^{(k)}$ assigned to cluster c, and α, a are parameters of a learned linear model. In contrast to learning a linear model, one may directly build a predictor using scores for clusters. That is, it is intuitive to combine the scores of clusters using a convex combination according to the frequency of clusters in a sample-set: $\hat{f}_s(\hat{\mathcal{X}}^{(k)}) = \sum_{c=1}^{C} \hat{r}_c^{(k)} s_c$, where s_c is the (pre-trained) score of the c-th region as described above, which is acting as a linear coefficient in the predictor $\hat{f}_s$. I.e. if we have scores for each cluster (coming from a pre-trained CKME model), then weighing these scores according to their respective prevalence in sample-sets should be predictive. Note that although this line of reasoning is *semantically intuitive*, the CKME cell scores were not trained to be used in this fashion, hence there is no guarantee that $\hat{f}_s$ will be predictive. We compare the accuracy of learning a linear model over cluster frequency features (denoted as *Linear*), $\hat{f}$, to directly predicting using fixed cluster scores, f_s, in Table 4.4. We consider cluster scores coming directly from respective centroids $s_c = s(\nu_c) = \varphi(\nu_c)^T\beta + b$ as *Centroid CKME*, and cluster scores coming from averages $s_c = s(G_c) = \frac{1}{|G_c|}\sum_{x\in G_c}\varphi(x)^T\beta + b$ as *Average CKME*. As an additional baseline, we also considered building a predictor through cell scores derived from MELD [19]. MELD is an alternative method that computes a score for every cell by first estimating the probability density of each sample along a kNN graph and then computes the relative likelihood of observing a cell in one experimental condition relative to the rest. As MELD cannot directly assign scores to post-hoc centroids, we utilize the average MELD scores for prediction $s_c = \frac{1}{|G_c|}\sum_{x\in G_c} s_x^{\text{MELD}}$, denoted as *Average MELD*. As shown in Table 4.4, predictors based on both CKME cluster scores (*Centroid CKME* and *Average CKME*) perform comparably to learning a linear model (*Linear*). Here we see that these scores are semantically meaningful since they retain intuitive predictive properties after manipulations. Again, this is surprising given that our CKME cell scores were not trained for this purpose (for prediction with cluster frequencies). This point is further emphasized by the lesser predictive performance of scores from MELD (*Average MELD*), which also seeks to provide semantically meaningful scores and was not trained for prediction in this fashion.

4.8.2 Biological Validation

An important advantage of interpretable models is their capacity to uncover scientific knowledge [23]. Therefore, we would like to know whether the patterns automatically learned by CKME are consistent with existing knowledge discovered independently via manual analysis. To do so, we focus our analysis on a salient cluster in the Preeclampsia dataset. More specifically,

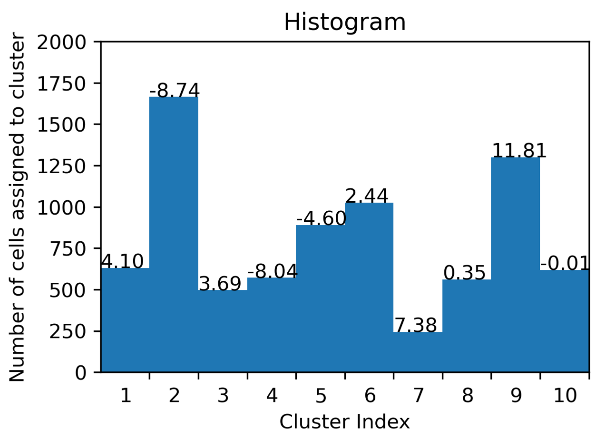


Figure 4.6: We computed the frequency of (e.g. number of) cells assigned to each of 10 clusters, or cell-populations. We further computed the score of corresponding centroids (value shown above each bar).

we used our scores to prioritize cells that were different between control (healthy) and preeclamptic women. Cells across sample-sets were clustered into one of 10 clusters. We then computed the average score of the cells assigned to each cluster. We first identified cell-populations associated with patient phenotype using CKME scores for each cluster, followed by a finer per-cell analysis highlighting clinically-predictive individual cells and lastly identifying their defining features.

4.8.2.1 Identifying Phenotype-Associated Cell-Populations

To link particular clusters (e.g. cell-populations) to clinical phenotype, we first prioritized cluster 2, based on its highly negative score according to CKME (-8.74, see Fig. 4.6). A highly negative score implies that it likely contained a large number of cells predicted as “control”. The prominent protein markers expressed in cluster 2 were CD3, CD4, CD45RA, and MAPKAPK2 and indicated this is a cell-population of naive CD4$^+$ T cells expressing MAPKAPK2 (Fig. 4.7**a**). The negative score of cluster 2 aligns with previous work, which showed that women with preeclampsia exhibit a decrease in MAPKAPK2$^+$ naive CD4$^+$ T-cells during the course of pregnancy, while healthy, women exhibit an increase [68].

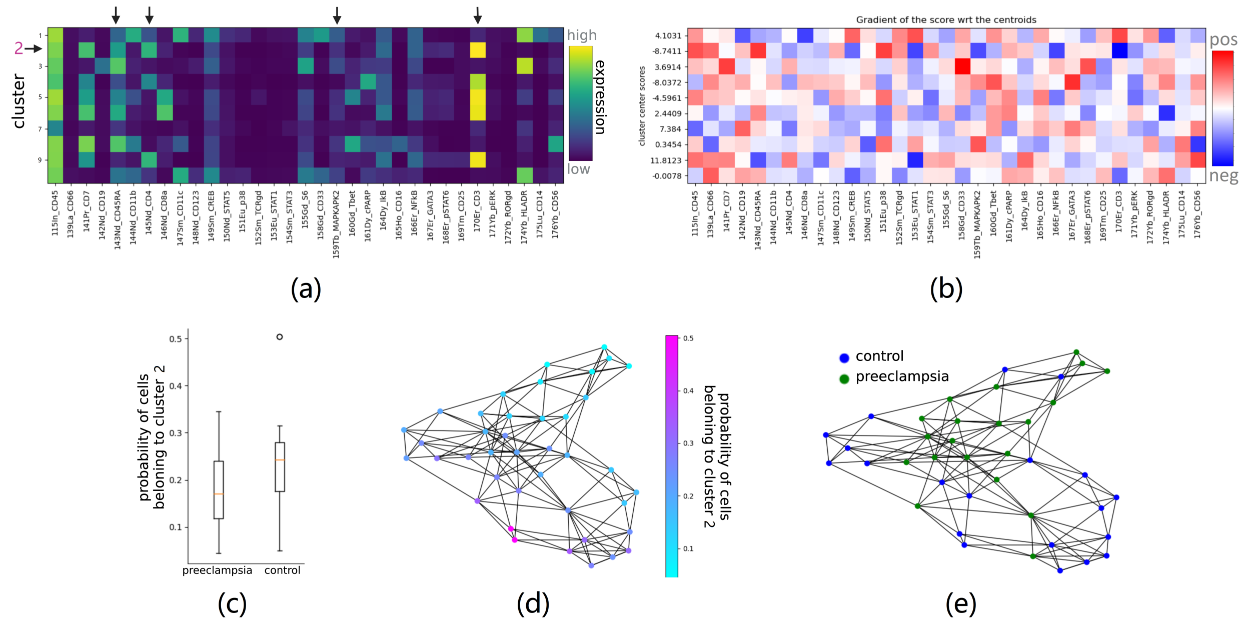


Figure 4.7: Our predicted scores prioritized cluster 2 (CD4$^+$ naive T-cells) as cell-population likely to have frequency differences between control and preeclamptic samples. (a) Cluster 2 was identified to correspond to a population of MAPKAPK2$^+$ CD4$^+$ naive T-cells according to protein markers (denoted with arrows). (b) Gradients of CKME scores for cluster centroids ν_c, $\nabla_x(\varphi(x)^T\beta)\,|_{x=\nu_c}$, where $\varphi(x)$ are random Fourier features and β are the learned model weights (eq. 4.8). (c) The distributions of frequencies of cells assigned to cluster 2 in sample-sets from preeclamptic and control women. (d) A k-NN graph connecting samples-sets (nodes) according to computed frequencies across cell-populations reveals a higher frequency of cells assigned to cluster 2 in control samples-sets. (e) The k-NN graph from (**d**), with each sample-set (node) colored by its ground-truth label.

We first compared the distributions of frequencies of cells assigned to cluster 2 (i.e. this population of naive CD4$^+$ T cells expressing MAPKAPK2) between sample-sets from preeclamptic and healthy control women (Fig. 4.7**c**). Consistent with our previous observations, sample-sets from control women had a statistically significantly higher proportion of cells assigned to cluster 2 in comparison to preeclamptic women (see Fig. 4.7**c** with a p-value of $p = 0.038$ under a Wilcoxon Rank Sum Test). As a complementary visualization, we constructed a k-nearest neighbor graph between sample-sets according to the computed frequencies across all cell-types (Fig. 4.7**d-e**). Here, each node represents a sample-set and an edge represents sufficient similarity between a pair of sample-sets according to the frequencies of cells across cell-populations. In Fig. 4.7**d**, sample-sets (nodes) are colored by the probability of their cells belonging to cluster 2. In

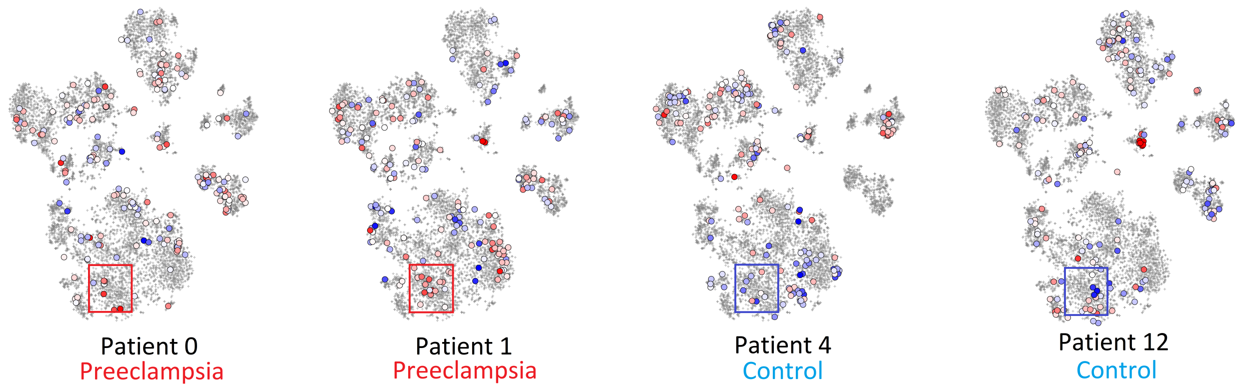


Figure 4.8: Visualizing computed per-cell CKME scores via tSNE. Cells in gray represent a general, patient-wide cellular landscape. Cells colored red (blue) imply high (low) associations with the preeclampsia phenotype.

comparison to sample-sets (nodes) colored by their ground-truth labels (Fig. 4.7**e**), we observed that control, healthy sample-sets such as the densely connected set of blue nodes in the bottom of Fig. 4.7**e** tend to have high frequencies of cells assigned to cluster 2.

4.8.2.2 tSNE Visualizations of Per-Cell CKME Scores in Patient Samples in the Preeclampsia CyTOF Dataset

Next, we perform a fine-grained analysis of individual cell scores. In contrast to previous approaches to identifying phenotype-associated cell-populations on a population level [181, 17], CKME provides much finer resolution and instead is able to highlight individual cells (in a digestible, synthesized KH subset) that are likely driving particular clinical phenotypes. For example, by combining cells from samples collected from healthy and preeclamptic women in the preeclampsia CyTOF dataset, we closely examined the patterns in the computed per-cell CKME scores and how they related to patient phenotypes. Briefly, 200 cells were sub-selected with Kernel Herding from each patient and the sub-selected cells from all patients are together projected into two dimensions with tSNE [190] to establish a general, patient-wide cellular landscape (gray cells in Fig. 4.8). For each patient, their sub-selected cells in the two-dimensional space are colored by their computed CKME score (Fig. 4.8). In particular, a cell colored red (blue) implies a high (low) association with the preeclampsia phenotype. In Fig. 4.8, we visualized scores for patients sampled from two preeclamptic patients (Patients 0 and 1) and two control patients (Patients 12 and 4)[9]. Remarkably, in the two preeclamptic women, we identified prominent subsets of red-colored nodes (outlined in red rectangles), implying a strong association with the preeclampsia phenotype. Based on the expression of phenotypic markers, these cells were identified as memory CD4+ T-cells (based on the expression of CD3, CD4, and CD45RA). In contrast, the memory CD4+ T-cells in the healthy patients exhibited low scores (blue-colored cells outlined in blue rectangles), indicating their negative association with the healthy phenotype. Taken together, the CKME scores highlighted the memory CD4+ T-cells as a key predictive population (also shown in previous work [68]). To further unravel the association between specialized immune cells and the preeclamptic and control patient phenotypes according to CKME scores, we investigated the prominent protein feature co-expression patterns of the subset of cells outlined in the rectangles in Fig. 4.8. Comparing the two preeclamptic (patients 0 and 1) to the two control (patients 4 and 12) sample-sets (Fig. 4.8), we quickly identified further nuanced cellular subsets associated with the cells with high (red) and low (blue) scoring CKME scores. A particular differentiating protein marker between the preeclamptic and control patients in this memory CD4$^+$ T-cell population was p38. The distribution of its expression in cells in the rectangular regions is shown in Fig. 4.9 and reveals prominent differences between the preeclamptic and control patients. p38 is a functional protein marker, indicating likely differences in signaling responses between control and preeclamptic samples. Interestingly, previous work [68] also independently showed that the expression of p38 in memory CD4$^+$ T-cells was an important feature for predicting preeclampsia status.

4.8.2.3 Assessing Protein Features in Preeclamptic and Healthy Patients

Lastly, we investigated the protein feature coexpression patterns of immune cell types from preeclamptic and healthy patients to further determine whether CKME scores are consistent with existing knowledge and could be used for hypothesis generation. To identify the cell populations

[9]These patients were from a testing set and were not used to learn the CKME scores.

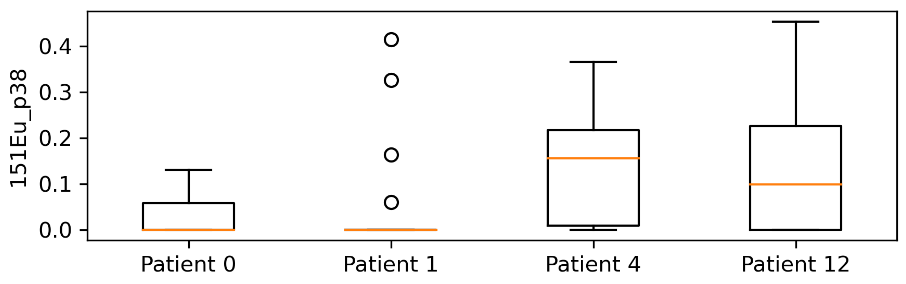


Figure 4.9: Distributions of features p38 for the cells inside the rectangles of Fig. 4.8.

associated with extremely positive or negative average CKME scores, we further annotated the clusters according to the expression of known protein markers (Fig. 4.7**a**). We found that the cell types that were strongly associated with the preeclampsia phenotype (positive average CKME score) were clusters 1 and 9, corresponding to classical monocytes and memory CD4$^+$ T cells (Table 4.5). In contrast, the cell types associated with the control phenotype (negative average CKME score) were clusters 2, 4, and 5 corresponding to naive CD4$^+$ T cells expressing MAP-KAPK2, naive CD4$^+$ T cells, and naive CD8$^+$ T cells (Table 4.5). Notably, this is consistent with previous work that found that the absolute monocyte count was significantly higher in preeclampsia patients than in controls [198, 195]. Moreover as highlighted previously, an abundance of naive CD4$^+$ T cells (Fig. 4.7**c-e**) or memory CD4$^+$ T cells (Fig. 4.8) are strong indicators of healthy or preeclampsia status, respectively [24, 37].

We also provide an alternative investigation of important protein markers in CKME scores. In particular, we study how minuscule changes in the expression of the protein markers defining the cluster (cell type) would increase the CKME score and thus have a stronger association with the clinical phenotype. To do so, we computed the gradient of the CKME score with respect to the cluster centroid. I.e. we compute $\nabla_x(\varphi(x)^T\beta)\,|_{x=\nu_c}$ for cluster centroids ν_c, , where $\varphi(x)$ are random Fourier features and β are the learned model weights (eq. 4.8). Doing so will uncover what small changes in protein markers alter the CKME of cluster centers, which serves as one proxy for feature importance.

Table 4.5: Gradient analysis of each cell-population highlights prominent features in clinically-associated cell-populations.

ID/CKME score	cell type (expression)	gradient	cell type (gradient)
1/4.1031	classical monocytes	$\downarrow$ CD14 $\uparrow$ CD16 $\uparrow$ STAT1	nonclassical monocytes STAT1$^+$
9/11.8123	memory CD4$^+$ T cells	$\uparrow$ CD4 $\downarrow$ CD45RA $\uparrow$ STAT3	memory CD4$^+$ STAT3$^+$ T cells
2/-8.7411	naive CD4$^+$ MAPKAPK2$^+$ T cells	$\uparrow$ CD45RA $\downarrow$ MAPKAPK2 $\uparrow$ p38	naive CD4$^+$ p38$^+$ T cells
4/-8.0372	naive CD4$^+$ T cells	$\uparrow$ Tbet $\uparrow$ GATA3	CD4$^+$ Tbet$^+$ GATA3$^+$ T cells
5/-4.5961	naive CD8$^+$ Tbet$^+$ T cells	$\uparrow$ Tbet $\uparrow$ p38	naive CD8$^+$ Tbet$^+$ p38$^+$ T cells

The gradient heatmap (as shown in Fig. 4.7**b**) provides a value for each feature within a centroid, where the corresponding direction of change specifies whether an increase (positive) or decrease (negative) in protein expression would increase the association with the preeclampsia phenotype. By prioritizing large magnitude changes in particular features that define cell types, in addition to signaling markers, we observed a shift in the expression of prioritized proteins. When examining the gradient feature changes in cluster 1 (generally defined as classical monocytes by the expression of CD14$^+$ CD16$^-$), we observed a decrease in CD14 and an increase in CD16 expression (Fig. 4.7**a-b**, Table 4.5). This indicates that a transition to a nonclassical monocyte phenotype (CD14$^{\text{med}}$ CD16$^{\text{med}}$) may cause a sample-set to appear more preeclamptic. Interestingly, previous work has indicated that the subpopulation composition of monocytes (classical, intermediate, nonclassical) significantly varies between preeclamptic and control patients [195]. With respect to cluster 2 (naive CD4$^+$ T cell population expressing MAPKAPK2), we observed a decrease in MAPKAPK2 and an increase in p38 expression (Fig. 4.7**a-b**, Table 4.5). This further highlights that p38 may be a good functional marker for distinguishing between healthy and preeclamptic patients in both naïve and memory CD4$^+$ T cells.

4.9 Closing Remarks

In this work, we introduced CKME (Cell Kernel Mean Embedding) as a method to link cellular heterogeneity in the immune system to clinical or external variables of interest, while simultaneously facilitating biological interpretability. As high-throughput single-cell immune profiling techniques are being readily applied in clinical settings [59, 68, 49], and there are critical needs to 1) accurately diagnose or predict a patient's future clinical outcome and 2) to explain the particular cell-types driving these predictions. Recent bioinformatic approaches have uncovered [17, 181, 79] or learned [5, 221] immunological features that can accurately predict a patient's clinical outcome. Unfortunately, these existing methods must often make a compromise between interpretability (e.g. simpler, more transparent methods, such as [79, 5] and accuracy (e.g. more complicated, opaque methods, such as [221]). Our experimental results show that CKME allows for the best of both worlds, yielding state-of-the-art accuracies, with a simple, more transparent model.

We leveraged the transparency of CKME with a thorough analysis. First, we showed that cell scores stemming from CKME retain intuitive desirable properties for cell contributions. We studied the semantics of CKME cell scores in tasks that CKME *was not explicitly trained for* and found that CKME cell scores were useful for those scenarios. This suggests that, through simple supervised training, CKME can be used to obtain insights into a myriad of analyses that it was not trained for. Furthermore, cell-population, individual-cell, and protein feature coexpression analyses yielded insights that aligned with previous literature (e.g. [68]). This suggests that CKME is a robust approach to uncovering scientific knowledge.

In summary, CKME enables a more comprehensive, automated analysis and interpretation of multi-patient flow and mass cytometry datasets and will accelerate the understanding of how immunological dysregulation affects particular clinical phenotypes. Future directions may include alternate synthesizing approaches to Kernel Herding and variants that weigh cells differently. Furthermore, in addition to the gradient-based analysis presented in Sec. 4.8.2.3, we shall explore other methodologies for assessing feature importance in CKME scores such as fitting local, sparse models [158].

CHAPTER 5: LEVERAGING RELATED POINTS IN A SEQUENCE FOR TIME SERIES IMPUTATION

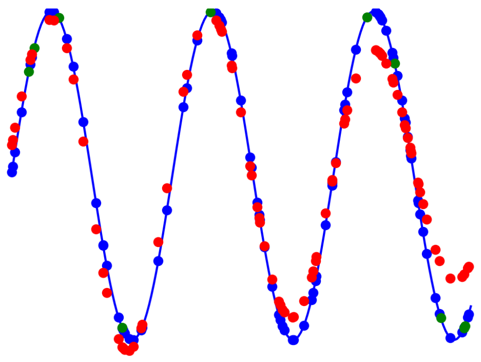
5.1 Notation

- $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$: a set with the set elements $\mathbf{x}_i \in \mathcal{X}$, where $\mathcal{X}$ represents the domain of each set element.
- $\mathbf{S} = \{\mathbf{s}_i\}_{i=1}^N$: a time series with N observations. Each observation $\mathbf{s}_i$ is a tuple $(t_i, \mathbf{x}_i)$, where $t_i \in \mathbb{R}^+$ denotes the observation time and $\mathbf{x}_i \in \mathbb{R}^d$ represents the observed data.
- $\hat{\mathbf{S}} = \{\hat{\mathbf{s}}_j\}_{j=1}^M$: data to impute as a set. M is the number of missing time points. Each set element $\hat{\mathbf{s}}_j$ is a tuple $(\hat{t}_j, \Delta\hat{t}_j)$

5.2 Introduction

Similar to the previous chapter, here we consider the case of time series data, where the input is also a collection. A time series is a collection of feature vectors indexed at different times. We consider the time series imputation problem where we need to predict the features at the missing times given the features at the observed times. We view these observed features as a collection of given related instances and develop an algorithm to exploit these related instances to predict features at the missing times.

Missing values are common in real-world time series, e.g. traffic trajectories often contain missing data due to unreliable sensors or object occlusion [210, 200]. Recovering those missing values is useful for the downstream analysis of time series. Modern approaches impute missing data in a data-driven fashion. For example, recurrent neural networks (RNNs) are applied in [127, 27, 22, 107], methods built on Neural Ordinary Differential Equations (NODEs) [29] are proposed in [162, 39, 89], and a family of models called Neural Process [62, 90] that learns a distribution over functions based on the observed data could also be leveraged.


(a) Latent-ODE

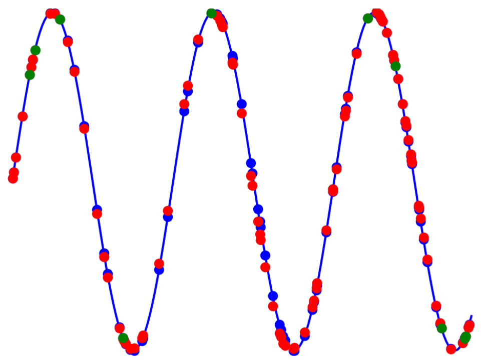

(b) NRTSI

Figure 5.1: Imputation results of Latent-ODE and NRTSI on a sinusoidal dataset. Red points are the imputed data, green points are the observed data and the blue points are the ground-truth data. We can see that Latent-ODE suffers from the error compounding problem as the imputation error accumulates through time while our NRTSI does not.

However, these existing works all have their own deficiencies. Models that are built on RNNs or NODEs usually employ a sequential imputation order, meaning that the imputed data x_t at timestep t is predicted based on the already imputed data x_{t-1} at the previous timestamp $t-1$. Since x_{t-1} inevitably contains errors, x_t is even more inaccurate and the errors will accumulate through time, resulting in poor long-horizon imputations for time series that are sparsely observed. This problem is known as *error compounding* in the fields of time series analysis [192, 218, 128, 129] and reinforcement learning [6, 113, 28, 25]. We illustrate this problem of a NODE-based method called Latent-ODE [162] in Fig 5.1 (a). Previous work [123] alleviates this problem with an RNN-based model, NAOMI, that imputes from coarse to fine-grained resolutions in a hierarchical imputation order. However, we find the hierarchical strategy in NAOMI is sub-optimal due to its reliance on RNNs. Neural Process models [61, 90, 62] do not impose any recurrent structures. However, they impute all the missing data at once without exploiting the hierarchical information of temporal data, which we find deteriorate their performance.

In this work, we propose NRTSI, a **N**on-**R**ecurrent **T**ime **S**eries **I**mputation model. One of our key insights is that when imputing missing values in time series, the valuable information from the observed data is *what happened and when*. This information is most naturally represented as a set of (time, data) tuples. We propose a novel imputation model to leverage observed data as a set of (time, data) tuple instances and impute the unobserved missing data. This is in stark contrast to previous works (e.g. NAOMI [123]) where observed data are leveraged recurrently so that the temporal information (when things happened) is unnecessarily entangled with the order of points being processed. Our natural set representation not only disentangles the data processing order from the temporal information, but also enables us to design an hierarchical imputation strategy that is efficient and principled. To the best of our knowledge, we are the first to jointly leverage the set formulation of time series and hierarchical imputation. We achieve the new SOTA across multiple benchmarks. Without the set formulation, we have to use the inferior hierarchical algorithm in [123] due to the RNN sequential constraints; without the hierarchical formulation, we find directly using set formulation (e.g. [74, 61, 90, 62]) leads to much worse imputation performance. Despite its simplicity, we find that NRTSI effectively alleviates the problems of the existing methods in a single framework while achieving state-of-the-art performance across multiple benchmarks.

Our contributions are as follows:

- We reinterpret time series as a set of (time, data) tuples and propose a time series imputation approach, NRTSI, using permutation equivariant models.
- We propose an effective hierarchical imputation strategy that takes advantage of the non-recurrent nature of NRTSI and imputes data in a multiresolution fashion.
- We show that NRTSI can flexibly handle irregularly-sampled data, data with partially observed time dimensions, and perform stochastic imputations for non-deterministic time series.
- We perform experiments on a wide range of datasets to demonstrate state-of-the-art imputation performance of NRTSI compared to several strong baselines.

5.3 Problem Formulation

5.3.1 Motivation

Models such as RNNs that scan sequentially, at first glance, are natural fits for time series data due to their temporal nature. For imputation, however, these models typically suffer from a problem called *error compounding* as discussed above. To remedy these deficiencies, we reinterpret time series as a set of (time, data) tuples. The set formulation allows us to conveniently develop a hierarchical scheme that reduces the number of imputation steps required compared to the sequential scheme and thus effectively alleviates the error compounding problem. It also directly enables imputing irregularly sampled time points, since the set can contain tuples for arbitrary time points. *Note that since the time information is provided in the (time, data) tuples, the sequential order of the time series is not lost, and we can easily transform the set back to a sequence.*

5.3.2 Formulation

Throughout the paper, we denote a set as $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ with set elements $\mathbf{x}_i \in \mathcal{X}$, where $\mathcal{X}$ represents the domain of each set element. We denote a time series with N observations as a set $\mathbf{S} = \{\mathbf{s}_i\}_{i=1}^{N}$, where each observation $\mathbf{s}_i$ is a tuple $(t_i, \mathbf{x}_i)$, where $t_i \in \mathbb{R}^+$ denotes the observation time and $\mathbf{x}_i \in \mathbb{R}^d$ represents the observed data. Given an observed time series $\mathbf{S}$, we aim to impute the missing data based on $\mathbf{S}$. We also organize data to impute as a set $\hat{\mathbf{S}} = \{\hat{\mathbf{s}}_j\}_{j=1}^{M}$, where M is the number of missing time points. Each set element $\hat{\mathbf{s}}_j$ is a tuple $(\hat{t}_j, \Delta\hat{t}_j)$, where $\hat{t}_j \in \mathbb{R}^+$ is a timestep to impute and $\Delta\hat{t}_j \in \mathbb{R}^+$ denotes the missing gap (i.e. the time interval length between $\hat{t}_j$ and its closest observation time in $\mathbf{S}$). Formally, $\Delta\hat{t}_j$ is defined as $\Delta\hat{t}_j = \min_{(t_i,\mathbf{x}_i)\in\mathbf{S}} |t_i - \hat{t}_j|$. Note that both $\hat{t}_j$ and t_i can be real-valued scalars rather than fixed grid points, which enable NRTSI to handle irregularly-sampled timesteps. The missing gap $\Delta\hat{t}_j$ is essential for our hierarchical imputation procedure. As will be discussed in Sec 5.5.1, we select a subset $\mathbf{G} \subseteq \hat{\mathbf{S}}$ to impute at each hierarchy level based on the missing gap of the target time points.

The imputation results are denoted as $\mathbf{H} = \{\mathbf{h}_j\}_{j=1}^{|\mathbf{G}|}$ with $\mathbf{h}_j \in \mathbb{R}^d$, where $\mathbf{H}$ is predicted using an imputation model f as $\mathbf{H} = f(\mathbf{G}; \mathbf{S})$.

5.4 Background

Our hierarchical imputation procedure is inspired by the successful idea of multi-scale modeling of images [2, 88] and waveforms [189]. For example, image pyramid methods [2] represent visual patterns across multiple resolution scales to extract both global and local information. Also, the multi-resolution method is used for image synthesis. PG-GAN [88] improves image synthesis quality and stability by starting with synthesizing low-resolution images first and gradually activating more neural network layers to synthesize higher resolution images based on the prior information of low-resolution images. WaveNet is a pioneering work for autoregressive audio synthesis. In order to capture both the local and global information of the previously generated waveforms, several dilated convolution layers with different dilation rates is used.

5.5 Methods

5.5.1 Hierarchical Imputation

In this section, we introduce our proposed hierarchical imputation procedure given the imputation models f at each hierarchy level. We defer the details about the imputation models to Sec. 5.5.2.

Generative models have benefited from exploiting the hierarchical structure of data [88, 121]. Here, we propose to leverage a multi-resolution procedure for time series imputation. Specifically, we divide the missing time points into several hierarchy levels using their missing gaps (i.e. the closest distance to an observed time point). Intuitively, missing data that are far from the observed data are more difficult to impute. According to their missing gaps, we can either impute from small gap time points to large gap ones or vice versa. Empirically, we find starting from large missing gaps works better (as also indicated by [123]). Given the imputed values at the

Algorithm 3 IMPUTATION PROCEDURE

```
Require: $f_\theta^l$: imputation model at resolution level $l$; $L$: the maximum resolution level; $\mathbf{S}$: observed data; $\hat{\mathbf{S}}$: data to impute
1:  Initialize $\mathbf{G} \leftarrow \emptyset, l \leftarrow 0$
2:  while $l \leq L$ do
3:      $\hat{\mathbf{S}}^l \leftarrow \{(\hat{t}_j, \Delta\hat{t}_j) \mid (\hat{t}_j, \Delta\hat{t}_j) \in \hat{\mathbf{S}}, \lfloor 2^{L-l-1} \rfloor < \Delta\hat{t}_j \leq 2^{L-l}\}$
4:      while $\hat{\mathbf{S}}^l$ is not empty do
5:          $\mathbf{G} \leftarrow \{(\hat{t}_j, \Delta\hat{t}_j) \mid (\hat{t}_j, \Delta\hat{t}_j) \in \hat{\mathbf{S}}^l, \Delta\hat{t}_j = \max_j \Delta\hat{t}_j\}$
6:          $\mathbf{H} \leftarrow f_\theta^l(\mathbf{G};\ \mathbf{S})$
7:          $\mathbf{S} \leftarrow \mathbf{S} \cup \mathbf{H}$, $\hat{\mathbf{S}} \leftarrow \hat{\mathbf{S}} \setminus \mathbf{G}$, $\hat{\mathbf{S}}^l \leftarrow \hat{\mathbf{S}}^l \setminus \mathbf{G}$
8:          Update the missing gap information in $\hat{\mathbf{S}}$ and $\hat{\mathbf{S}}^l$
9:      end while
10:     $l \leftarrow l + 1$
11: end while
12: return Imputation result $\mathbf{S}$
```

current hierarchy level, the imputation at the higher hierarchy level will depend on those values. Note that the hierarchical imputation inevitably introduces some recurrent dependencies among missing time points, but since the number of hierarchy levels is typically much smaller than the number of missing time points, *the error compounding problem of NRTSI is not as severe as the sequential models.* To further reduce the imputation error at each level, we utilize a separate imputation model f_θ^l for each level l. The imputation model takes in all the observed data and the already imputed data at lower hierarchy levels as a set to impute the missing data at the current level. At each hierarchy level, the missing time points are imputed in descending order of their missing gaps. Please refer to Algorithm 3 for the proposed imputation procedure. We illustrate the imputation procedure in Fig 5.2 where at each hierarchy level NRTSI can impute multiple missing points in parallel thanks to the set representation of time series. Note that NRTSI can also handle irregularly-sampled time series, though we only show the procedure on regularly sampled ones in Fig 5.2 for convenience.

Similar to our hierarchical imputation procedure, NAOMI [123] also explore the multiresolution structure of time series. NAOMI first encodes the observed data using a bidirectional RNN. Then the missing data are imputed by several multiresolution decoders followed by updating the RNN hidden state to take the newly imputed data into account. To make the hidden state updates

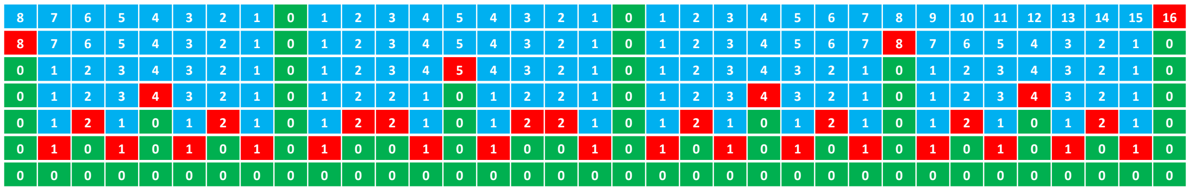


Figure 5.2: Illustration of the imputation procedure. Blue, green and red boxes respectively represent missing data, observed data, and data to impute next. Numbers inside each box represent the missing gap to the closest observed data and we assume the missing gap of observed data to be 0. When deciding which data to impute next, we always select the group of data with the largest missing gap (Line 5 Algorithm 3). Given a time series with 2 observed values and 32 missing values, the imputation procedure starts at the first row and ends at the bottom row where all data are imputed.

efficient, NAOMI chooses to first impute the data with the largest missing gap located between the two earliest observed time points. Therefore, the first imputed data might not have the largest missing map throughout the whole time series. In contrast, NRTSI always imputes the data with the global largest missing gap first thanks to our set formulation. Also, NAOMI only imputes a single time point at each step, while NRTSI can impute multiple time points together.

5.5.2 Imputation Model f_θ^l

In this section, we describe the imputation model f_θ^l used at each hierarchy level l. The model takes in a set of known time points $\mathbf{S}$ (either observed or previously imputed at lower hierarchy levels) and imputes the values for a set of target time points $\mathbf{G}$. To respect the unordered nature of sets, the imputation model f_θ^l is designed to be permutation equivariant w.r.t. $\mathbf{G}$ and permutation invariant w.r.t. $\mathbf{S}$, i.e. $\rho(\mathbf{H}) = f_\theta^l(\rho(\mathbf{G}); \pi(\mathbf{S}))$, where π and ρ represent two arbitrary permutations. Theoretically, any permutation equivariant architecture can be seamlessly plugged in as long as we use a permutation invariant embedding on $\mathbf{S}$. Representative architectures include DeepSets [223], ExNODE [114] and Transformers [191, 102]. In this work, we adopt the multi-head self-attention mechanism in Transformers for its established strong capability of modeling high-order interactions.

Time Encoding Function Transformers do not require the sequences to be processed in sequential order. Originally, the input to Transformers is a set of word embeddings and positional embeddings. Although the input is an unordered set, the positions of word embeddings are informed by their corresponding positional embeddings. Similar to the positional embedding in Transformers, we embed the time using a function ϕ to transform a time stamp $t \in \mathbb{R}^+$ to a higher dimensional vector $z = \phi(t) \in \mathbb{R}^\tau$, where

$$z_{2k}(t) := \sin\left(\frac{t}{\nu^{2k/\tau}}\right) \qquad z_{2k+1}(t) := \cos\left(\frac{t}{\nu^{2k/\tau}}\right) \tag{5.1}$$

with $k \in \{0, \ldots, \tau/2\}$, τ denoting the dimensionality of the time embedding, and ν representing the expected maximum time scale for a given data set. This time embedding function is first proposed in [74].

Implementation At each hierarchy level l, a subset of missing time points $\mathbf{G}$ are first selected based on their missing gaps $\Delta\hat{t}_j$, then the imputation model f_θ^l imputes the missing values by $\mathbf{H} = f_\theta^l(\mathbf{G}; \mathbf{S})$, where $\mathbf{S} = \{(\phi(t_i), \mathbf{x}_i)\}$ and $\mathbf{G} = \{\phi(\hat{t}_j)\}$. Note that $\Delta\hat{t}_j$ are ignored here since they are only used to define the hierarchy levels (see Algorithm 3). The elements in $\mathbf{S}$ and $\mathbf{G}$ are transformed to tensors by concatenating the data $\mathbf{x} \in \mathbb{R}^d$ and the time encoding vector $\phi(t) \in \mathbb{R}^\tau$ computed via (5.1). Since the elements in $\mathbf{G}$ do not contain $\mathbf{x}$, we use d-dimensional zero vectors $\mathbf{0} \in \mathbb{R}^d$ as placeholders. We also add a binary scalar indicator to distinguish missing values and observed values. That is,

$$\begin{aligned} \mathbf{s}_i = (\phi(t_i), \mathbf{x}_i) \in \mathbf{S} \quad &\rightarrow \quad \mathbf{s}_i = [\phi(t_i), \mathbf{x}_i, 1] \in \mathbb{R}^{\tau+d+1}, \\ \mathbf{g}_j = \phi(\hat{t}_j) \in \mathbf{G} \quad &\rightarrow \quad \mathbf{g}_j = [\phi(\hat{t}_j), \mathbf{0}, 0] \in \mathbb{R}^{\tau+d+1}, \end{aligned} \tag{5.2}$$

where $[\cdot]$ represents the concatenation operation. Now that the elements in $\mathbf{S}$ and $\mathbf{G}$ are all transformed to vectors with same dimensionality, we can combine them into one set and pass it through the permutation equivariant imputation model f_θ^l, i.e. $\mathbf{H} = f_\theta^l(\mathbf{S}; \mathbf{G})$. Specifically, we implement f_θ^l by the following steps:

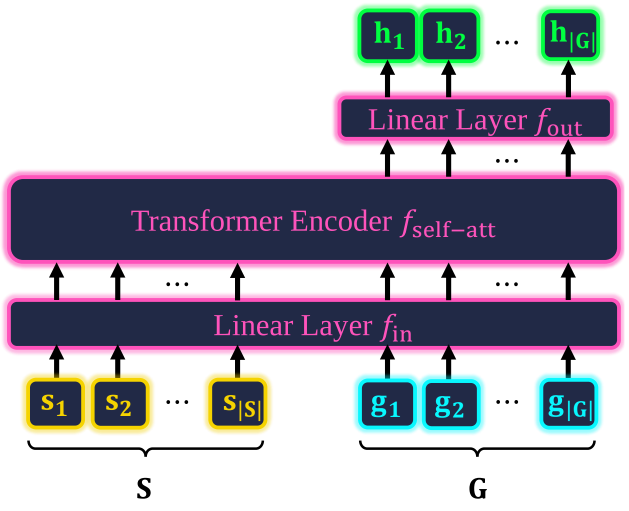


Figure 5.3: Imputation model.

$$
\begin{aligned}
\mathbf{S}^{(1)} \cup \mathbf{G}^{(1)} &= f_{\text{in}}(\mathbf{S} \cup \mathbf{G}) \\
\mathbf{S}^{(2)} \cup \mathbf{G}^{(2)} &= f_{\text{enc}}(\mathbf{S}^{(1)} \cup \mathbf{G}^{(1)}) \qquad (5.3) \\
\mathbf{H} &= f_{\text{out}}(\mathbf{G}^{(2)}).
\end{aligned}
$$

At the first step, a linear layer $f_{\text{in}} : \mathbb{R}^{\tau+d+1} \to \mathbb{R}^{d_h}$ maps the input data to a high-dimensional space in a point by point fashion. Then, a permutation equivariant Transformer encoder $f_{\text{enc}} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h}$ is used to model the interactions between $\mathbf{S}^{(1)}$ and $\mathbf{G}^{(1)}$. The Transformer encoder is composed of multiple alternating multi-head self-attention layers and feedforward layers, allowing the elements in $\mathbf{S}^{(1)}$ and $\mathbf{G}^{(1)}$ to effectively exchange information, i.e. $\mathbf{G}^{(1)}$ can attend to $\mathbf{S}^{(1)}$ to gather the observed information and $\mathbf{S}^{(1)}$ can be informed about what timestamps to impute by attending to $\mathbf{G}^{(1)}$. Finally, the imputation results $\mathbf{H}$ are obtained via another linear layer $f_{\text{out}} : \mathbb{R}^{d_h} \to \mathbb{R}^{d}$ on $\mathbf{G}^{(2)}$. The architecture of the imputation model is illustrated in Fig 5.3. Since the set elements are processed individually by the linear layers and the attention mechanism [191, 133, 170] in f_{enc} is permutation equivariant, the overall model f_θ^l is permutation equivariant w.r.t. $\mathbf{G}$ and permutation invariant w.r.t. $\mathbf{S}$.

Training Objective We denote our imputation model with learnable parameters θ at level l as f_θ^l, which include the two linear layers and the Transformer encoder. The optimization objective is

$$\min_\theta \mathbb{E}_{\mathbf{G}\sim p(\mathbf{G}),\mathbf{S}\sim p(\mathbf{S}),\mathbf{Y}\sim p(\mathbf{Y})}\left[\frac{1}{|\mathbf{G}|}\sum_{j=1}^{|\mathbf{G}|}\mathcal{L}(\mathbf{h}_j,\mathbf{y}_j)\right], \tag{5.4}$$

where $\mathbf{h}_j \in \mathbf{H} = f_\theta^l(\mathbf{G};\mathbf{S})$ is an imputed data and $\mathbf{y}_j \in \mathbf{Y}$ denotes the corresponding ground truth imputation target. For deterministic datasets, we use Mean Square Error (MSE), i.e. $\mathcal{L}(\mathbf{h}_j,\mathbf{y}_j) = ||\mathbf{h}_j - \mathbf{y}_j||_2^2$. For stochastic datasets (see Section 5.5.3), we minimize the negative log-likelihood of a Gaussian distribution with diagonal covariance, i.e.

$$\mathcal{L}(\mathbf{h}_j,\mathbf{y}_j) = -\log\mathcal{N}(\mathbf{y}_j|\mu(\mathbf{h}_j),\text{diag}(\sigma(\mathbf{h}_j))), \tag{5.5}$$

where $\mu : \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ are two linear mappings. We use a training procedure (Algorithm 4) similar to Algorithm 3. During training, we regard the ground truth imputation target $\mathbf{Y}$ as observed data for the subsequent training steps rather than the imputed $\mathbf{H}$. This resembles the Teacher Forcing algorithm [215] used to stabilize the training of RNNs.

5.5.3 Model Variants

Next, we show NRTSI is general enough to naturally handle irregularly sampled time series, stochastic time series, and partially observed time series in a unified framework.

Irregularly-sampled Time Series For regularly-sampled time series, there are typically multiple time points with the same missing gaps. For irregularly-sampled time series, however, missing time points tend to have unique missing gaps. Therefore, imputing from large missing gaps to small missing gaps will reduce to an autoregressive model, which could incur a high computation demand. Instead, we modify line 5 in Algorithm 3 by imputing time points with similar missing gaps together, i.e. $\mathbf{G} \leftarrow \{\ (\hat{t_j}, \hat{\Delta t_j}) \mid (\hat{t_j}, \hat{\Delta t_j}) \in \hat{\mathbf{S}}^l, \hat{\Delta t_j} \in (a, b]\}$ where $a = \max_j \hat{\Delta t_j} - \Delta$, $b = a + \Delta$, and $\Delta \in \mathbb{R}^+$ is a hyperparameter.

Stochastic Time Series Often one models only the mean time series for single imputation, we instead model multiple potential time series conditioned on the observed points for multiple imputations (see Fig 5.5). For deterministic datasets, we can impute the mean for data in $\mathbf{G}$ simultaneously as no sampling is required. For stochastic datasets, however, we need to sample from the distribution (5.5) and the samples may be incongruous if we sample for all the elements in $\mathbf{G}$ simultaneously. This is because the intermediate imputations should affect the conditional distribution of later imputed steps. To solve this problem, we propose to impute data with large missing gaps one by one. Based on the observation that missing data with small missing gaps are almost deterministic, they can be imputed simultaneously in parallel to avoid the high complexity of sampling sequentially. In practice, we find this modification does not significantly slow down the imputation process as there are much fewer data with large missing gaps compared to the ones with small missing gaps.

Algorithm 4 TRAINING PROCEDURE AT LEVEL l

```
Require: $f_\theta^l$: imputation model at resolution level $l$, $\mathbf{S}$: observed data, $\hat{\mathbf{S}}$: data to impute, $\mathbf{Y}$: ground-truth imputation target
 1: Initialize current imputation set $\mathbf{G}$ as an empty set, i.e. $\mathbf{G} \leftarrow \emptyset$
 2: if $l < L$ then
 3:     Initialize $\theta$ from the higher level trained model $f_\theta^{l+1}$
 4: else
 5:     Randomly initialize $\theta$
 6: end if
 7: while $f_\theta^l$ does not converge do
 8:     Sample a batch of training data $\hat{\mathbf{S}} \sim p(\hat{\mathbf{S}}), \mathbf{S} \sim p(\mathbf{S}), \mathbf{Y} \sim p(\mathbf{Y})$
 9:     Find times points to impute at current level $\hat{\mathbf{S}}^l \leftarrow \{ (\hat{t}_j, \Delta\hat{t}_j) \mid (\hat{t}_j, \Delta\hat{t}_j) \in \hat{\mathbf{S}}$ and $\lfloor 2^{L-l-1} \rfloor < \Delta\hat{t}_j \leq 2^{L-l} \}$
10:     while $\hat{\mathbf{S}}^l$ is not empty do
11:         Find the data to impute with the largest missing gap and put them into $\mathbf{G}$, i.e.
            $\mathbf{G} \leftarrow \{ (\hat{t}_j, \Delta\hat{t}_j) \mid (\hat{t}_j, \Delta\hat{t}_j) \in \hat{\mathbf{S}}^l, \Delta\hat{t}_j = \max_j \Delta\hat{t}_j \}$
12:         Compute the imputation results $\mathbf{H} = \{(\hat{t}_j, \mathbf{h}_j)\}_{j=1}^{|\mathbf{G}|}$ based on observed data $\mathbf{S}$, i.e. $\mathbf{H} \leftarrow f_\theta^l(\mathbf{G};\ \mathbf{S})$
13:         Train $\theta$ via the objective function in (5.4) in the main text
14:         Regard the ground-truth imputation target $\mathbf{Y}$ as observed points and delete $\mathbf{G}$ from $\hat{\mathbf{S}}^l$,
            i.e. $\mathbf{S} \leftarrow \mathbf{S} \cup \mathbf{Y}$, $\hat{\mathbf{S}}^l \leftarrow \hat{\mathbf{S}}^l \setminus \mathbf{G}$
15:     end while
16: end while
```

Partially Observed Time Series In practice, a timestep may be only partially observed, i.e. only a subset of features is missing at that time. Our hierarchical imputation procedure can be easily extended to this partially observed scenario. However, missing gaps may no longer be the driving factor for an effective imputation order, as the number of dimensions observed may affect the effectiveness of imputations. Therefore, we modify Algorithm 3 to impute the timesteps with the most missing dimensions first rather than the timesteps with the largest missing gap. We also modify the data representation to $\mathbf{s}_i = [\phi(t_i), \mathbf{x}_i, \mathbf{m}_i] \in \mathbb{R}^{d+\tau+d}$ where $\mathbf{m}_i \in \{0, 1\}^d$ is a binary mask indicating which dimensions are observed.

5.6 Related Work

Time Series Imputation Deep generative models offer a flexible framework for imputation. Several variants of RNNs [27, 22, 33, 154] are proposed to impute time series. Models based on NODEs [29], such as LatentODE [162], ODE-RNN [39] and NeuralCDE [89], are also proposed to impute irregularly-sampled data. Generative adversarial networks are leveraged in [51, 222, 126]. However, all of these works are recurrent.

NAOMI [123] performs time series imputation via a non-recurrent imputation procedure that imputes from coarse to fine-grained resolutions using a divide-and-conquer strategy. However, NAOMI relies on RNNs to process observed time points, which limits its application for irregularly-sampled time data and loses the opportunity to efficiently impute multiple time points in parallel. Moreover, the hierarchical imputation procedure of NAOMI assumes that the multivariate data at a timestep is either completely observed or completely missing along all the dimensions. In contrast, the imputation procedure of NRTSI also works well when dimensions are partially observed.

Set Formulation of Time Series Similar to NRTSI, SeFT [74], attentive neural process (ANP) [90], Conditional Score-based Diffusion Models (CSDI) [186], TTS-GAN [111], and TTS-CGAN [112] view a temporal sequence as an unordered set, where each set element is a tuple

Table 5.1: Quantitative comparison on Billiards dataset. Statistics closer to the expert indicate better performance. † results are taken from NAOMI [123].

Models	Linear†	kNN†	GRUI†	MaskGAN†	ANP	mTAN	CSDI	SingleRes†	NAOMI†	NRTSI	Expert
Sinuosity	1.121	1.469	1.859	1.095	1.364 ± 0.012	1.099 ± 0.010	1.231 ± 0.012	1.019	1.006	$\mathbf{1.003 \pm 0.002}$	1.000
step change ($\times 10^{-3}$)	**0.961**	24.59	28.19	15.35	18.95 ± 2.82	14.59 ± 2.17	14.92 ± 2.05	9.290	7.239	5.621 ± 0.752	1.588
reflection to wall	0.247	0.189	0.225	0.100	0.134 ± 0.013	0.091 ± 0.010	0.089 ± 0.011	0.038	0.023	$\mathbf{0.021 \pm 0.002}$	0.018
L2 loss ($\times 10^{-2}$)	19.00	5.381	20.57	1.830	3.762 ± 0.659	1.102 ± 0.316	1.115 ± 0.392	0.233	0.067	$\mathbf{0.024 \pm 0.003}$	0.000

that contains the observed data $\mathbf{x}_t \in \mathbb{R}^d$ at timestep $t \in \mathbb{R}$ and an encoding [217] of the time $\phi(t)$. SeFT [74] has shown that this set formulation is superior to several strong recurrent baselines for time series classification. However, only time series classification is considered in SeFT [74] and [217] and only time series synthesis is considered in TTS-GAN [111] and TTS-CGAN [112], while we propose a novel model that targets at time series imputation and allows effective information exchanges between observed data and missing data. Although ANP is applicable for the imputation task, the information of what timesteps to impute (target input) is not utilized when ANP uses self-attention to compute the representations of observed data (context input/output pairs). Therefore, the representation might be suboptimal. CDSA [134] performs time series imputation via self-attention without recurrent modules. However, CDSA is specifically designed for geo-tagged data. Multi-Time Attention Networks (mTAN) [175] learn an embedding of continuous time values and use an attention mechanism for interpolation and classification. However, mTAN still contains recurrent modules (e.g. bidirenctional RNN). Besides, ANP, CDSA, CSDI and mTAN do not exploit the multiresolution information of sequences, which may impact their imputation performance according to our ablation study in Section 5.7.5.1. Also,

5.7 Experiments

In this section, we evaluate NRTSI on popular time series imputation benchmarks against strong baselines. For fair comparisons, we use the same training/validation/testing splits for all the methods. Similar to masked self-supervised learning [209], *during training, we randomly mask out a subset of observed data and use the masked data as the ground truth imputation target to train models.* We use the same method to randomly mask out data for all the methods.

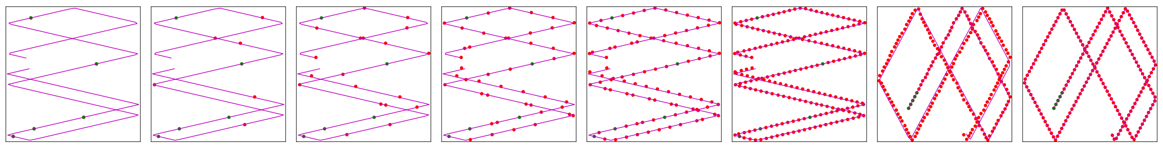

(a) Observed (b) Gap 16 (c) Gap 8 (d) Gap 4 (e) Gap 2 (f) Gap 1 (g) NAOMI (h) NRTSI

Figure 5.4: Imputation procedure on the Billiards dataset. The red points denote imputed data while the green points denote observed data. The purple solid line is the ground-truth trajectory. The initial observed data is shown in (a), the imputed data with missing gaps 16 to 1 are shown in (b)-(f). We omit the intermediate results at missing gaps 15, 7, 6, and 3 due to the limitation of space. In (g) and (h) we show the forward prediction results of NAOMI and NRTSI.

All the experiments conducted in this paper are repeated 5 times and we accordingly report the average metrics and their standard deviations. We defer the details about datasets, architectures, and training procedures to Section 5.7.4, 5.7.1, and 5.7.2.

Billiards Ball Trajectory Billiards dataset [164] contains regularly-sampled trajectories of Billiards balls in a rectangular world. Each ball is initialized with a random position and a random velocity and the trajectory is rolled out for 200 timesteps. All balls have a fixed size and uniform density, and friction is ignored.

On this dataset, we train and evaluate using MSE loss between imputed values and ground truth. We also report three additional metrics, *Sinuosity*, *step change* and *reflection to wall*, as reported in [123] to assess the realism of the imputed trajectories. We closely follow the setting in [123] and compare to all baselines mentioned there. Please refer to [123] for details about the baseline models. We also include ANP [90], mTAN [175] and CSDI [186] as baselines. Quantitative results are reported in Table 5.1, where `Expert` denotes the ground truth trajectories. Following [123], we randomly select 180 to 195 timesteps as missing for each trajectory and repeat the test set 100 times to make sure a variety of missing patterns are covered. Statistics that are closer to the expert ones are better. From Table 5.1, we can see NRTSI reduces the L_2 loss by 64% compared to NAOMI and compares favorably to baselines on all metrics except *step change*, as linear interpolation maintains a constant step size change by design, which matches with the underlying dynamics of this dataset. In Fig 5.4, we visualize the hierarchical imputation pro-

Table 5.2: Traffic data L_2 loss ($\times 10^{-4}$). † results are taken from NAOMI [123].

Linear†	GRUI†	kNN†	MaskGAN†	ANP	CSDI	mTAN	SingleRes†	NAOMI†	NRTSI
15.59	15.24	4.58	6.02	6.93 ± 1.91	5.89 ± 0.52	5.23 ± 0.44	4.51	3.54	$\mathbf{3.22 \pm 0.12}$

cedure on one example. Initially, only five points are observed (shown in green). Then, NRTSI imputes the time points with the largest missing gap (16) based on those observed points. It then hierarchically imputes the remaining missing time points with smaller missing gaps. The final imputed trajectory not only aligns well with the ground truth but also maintains a constant speed and straight lines between collisions. To better understand how NRTSI imputes, in Section 5.7.3, we visualize the attention maps of several attention heads and find that they capture different temporal relationships in accordance with previous findings [204].

In Fig 5.4 (g) and (h), we respectively show the forward prediction (predict the last 195 missing values based on the first 5 observed values) results of NAOMI and NRTSI. It can be seen that the trajectories predicted by NRTSI is more accurate and realistic compared to NAOMI, indicating the advantage of using a non-recurrent imputation model.

Traffic Time Series The PEMS-SF traffic [43] is a multivariate dataset with 963 dimensions at each time point, which represents the freeway occupancy rate from 963 sensors. The occupancy rate is regularly-sampled every 10 minutes throughout the day, resulting in the length of each time series being 144. Time series in this dataset is *non-stationary* [165] as statistical properties (e.g. mean of the occupancy rate) are not constant over time.

Similar to the Billiards experiment above, we train and evaluate using MSE loss. We also compare to the same set of baselines as the Billiards experiment. Following [123], we generate masked sequences with 122 to 140 missing values at random and repeat the testing set 100 times. The L_2 losses are reported in Table 5.2.

MuJoCo Physics Simulation MuJoCo is a physical simulation dataset created by [162] using the "Hopper" model from the Deepmind Control Suite [187]. Initial positions and velocities of

Table 5.3: MuJoCo dataset MSE loss (10^{-3}) comparison. † results are taken from [162].

Method	10%	20%	30%	50%
`RNN GRU-D`†	19.68	14.21	11.34	7.48
`ODE-RNN`†	16.47	12.09	9.86	6.65
`NeuralCDE`	13.52 ± 0.71	10.71 ± 0.57	8.35 ± 0.49	6.09 ± 0.41
`Latent-ODE`†	**3.60**	2.95	3.00	2.85
`ANP`	7.65 ± 0.47	4.37 ± 0.38	3.21 ± 0.36	2.97 ± 0.33
`CSDI`	6.64 ± 0.35	3.79 ± 0.37	2.96 ± 0.31	2.62 ± 0.32
`mTAN`	5.90 ± 0.45	3.17 ± 0.36	2.51 ± 0.32	2.35 ± 0.28
`NAOMI`	4.42 ± 0.41	2.32 ± 0.35	1.46 ± 0.13	0.93 ± 0.11
NRTSI	4.06 ± 0.38	**1.22 ± 0.11**	**0.63 ± 0.09**	**0.26 ± 0.02**

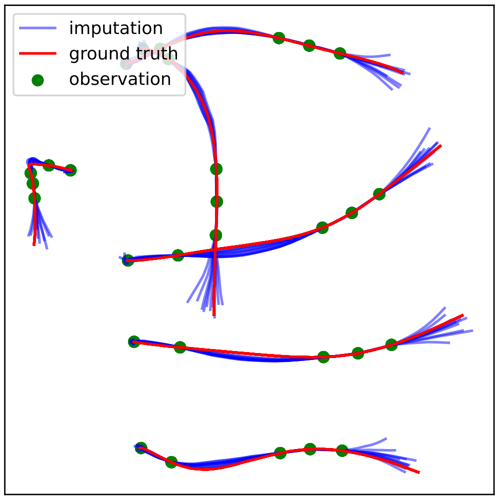
Table 5.4: Quantitative comparison on Football Player Trajectory. A larger *avg*MSE / *min*MSE indicate better diversity. Other statistics closer to the expert indicate better performance.

Models	`Latent-ODE`	`NAOMI`	`CSDI`	`ANP`	NRTSI	Expert
step change ($\times 10^{-3}$)	1.473±0.154	3.227±0.216	1.543±0.297	1.754±0.211	**2.401±0.087**	2.482
avg length	0.136±0.009	0.236±0.008	0.201±0.009	0.145±0.007	**0.175±0.004**	0.173
*min***MSE** ($\times 10^{-3}$)	19.53±1.44	4.079±0.487	8.142±1.005	6.652±0.881	**1.908±0.101**	0.000
*avg***MSE / *min*MSE**	1.16±0.09	1.12±0.07	1.53±0.07	1.19±0.10	**2.13± 0.08**	—
Imputation Time (s)	1.92	0.64	3.32	**0.43**	0.51	—

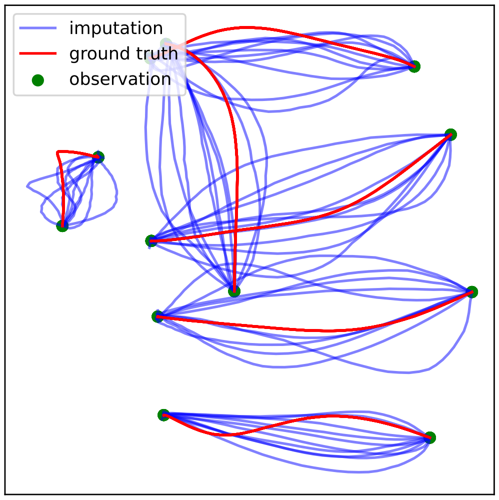
the hopper are randomly sampled such that the hopper rotates in the air and falls on the ground. The dataset is 14-dimensional.

MSE loss is used to train and evaluate NRTSI. Baseline models include Latent-ODE [162], ODE-RNN [162], GRU-D [27], NeuralCDE [89], ANP [90], mTAN [175], CSDI [186] and NAOMI [123]. We report the MSEs with different observation rates in Table 5.3. NRTSI compares favorably to all baselines with 20%, 30% and 50% observed data. When only 10% data are observed, NRTSI is comparable to Latent-ODE and NAOMI.

Football Player Trajectory This dataset is from the NFL Big Data Bowl 2021 [86], which contains the 2D trajectories of 6 offensive players and is therefore 12-dimensional. During training and testing, we treat all players in a trajectory equally and randomly permute their orders. Every time series contain 50 regularly-sampled time points with a sampling rate of 10 Hz.



(a) 5 timesteps observed



(b) 2 timesteps observed

Figure 5.5: Imputed trajectories of football players.

This dataset is stochastic and contains multiple modes since there could be many possible trajectories based on the sparsely observed data. Therefore, we follow [147] to use *min*MSE to evaluate the precision and the ratio between *avg*MSE and *min*MSE to evaluate the diversity of multiple imputed trajectories. *min*MSE and *avg*MSE respectively denote the minimum and the average MSEs between the ground truth and n independently sampled imputation trajectories. A small *min*MSE indicates that at least one of n samples has high precision, while a large *avg*MSE implies that these n samples are spread out. Similar to [123], we also use *average trajectory length* and *step change* to assess the quality of sampled trajectories.

For this dataset, we train NRTSI to minimize the negative log-likelihood as in (5.5). For each trajectory, we randomly select 40 to 49 timesteps as missing. According to the discussion in Sec 5.5.3, data with missing gaps larger than 4 are imputed one by one, while data with smaller missing gaps are imputed in parallel. We compare to several baselines such as Latent-ODE, NAOMI, CSDI and ANP that can impute stochastically. As shown in Table 5.4, NRTSI compares favorably to the baselines. The speed does not degrade too much by the one-by-one imputation on the large missing gap time points. It only takes 0.51 seconds to impute a batch of 64 trajectories on GTX 1080Ti, which is faster than NAOMI. Though ANP is faster than NRTSI, ANP does not

Table 5.5: Irregularly-sampled Billiards data L_2 loss ($\times 10^{-2}$).

Latent-ODE	NeuralCDE	ANP	mTAN	CSDI	NAOMI-Δ_t	NRTSI
19.48±1.64	34.01±1.99	29.31 ±1.53	3.542±0.447	3.823±0.521	1.121±0.265	**0.042±0.008**

impute hierarchically and has worse imputation quality. The *min*MSE and *avg*MSE scores are calculated using 10 independently imputed trajectories. As shown in Fig 5.5, the imputation is accurate around observed time points and captures the uncertainty for points far from the observations. When the observation is sparse, NRTSI can impute multiple realistic trajectories.

Irregularly-sampled Time Series We first test NRTSI on a toy synthetic dataset. Following Latent-ODE [162], we create an irregularly-sampled dataset containing 1,000 sinusoidal functions with 100 time points for each function, from which 90 timesteps are randomly selected as missing data. As shown in Fig. 5.1, both Latent-ODE and NRTSI capture the periodic pattern of this dataset, but NRTSI is more accurate and obtains a much lower MSE (7.17×10^{-4}) compared to Latent-ODE (5.05×10^{-2}).

Next, we evaluate NRTSI on an irregularly-sampled Billiards dataset. This dataset is created with the same parameters (e.g. initial ball speed range, travel time) as the regularly-sampled Billiards dataset. The only difference is that this dataset is irregularly-sampled. We compare NRTSI with two representative ODE-based approaches that can deal with irregularly-sampled data, namely Latent-ODE and NeuralCDE. We also modify NAOMI to handle irregularly-sampled data, which we call NAOMI-Δ_t. The time gap information between observations is provided to the RNN update function of NAOMI-Δ_t. We also compare to ANP, mTAN and CSDI which can handle irregularly-sampled data. In Table 5.5, we report the MSE losses between imputed and ground truth values. We randomly select 180 to 195 timesteps out of the total 200 timesteps as missing for each trajectory. According to Table 5.5, NRTSI outperforms the baselines by a large margin despite extensive hyperparameter search for these baselines. We show several trajectories imputed by NRTSI in Fig. 5.6. To further investigate the poor performance of Latent-ODE, NeuralCDE, and ANP, when the observation is dense (150 points observed), they all perform well.

However, they have difficulty in predicting the correct trajectories when the observation becomes sparse (e.g. with only 5 points observed). The excellent performance of NRTSI and NAOMI-Δ_t indicates the benefits of the multiresolution imputation procedure. Furthermore, the superiority of NRTSI over NAOMI-Δ_t demonstrates the advantage of the proposed set modeling approach.

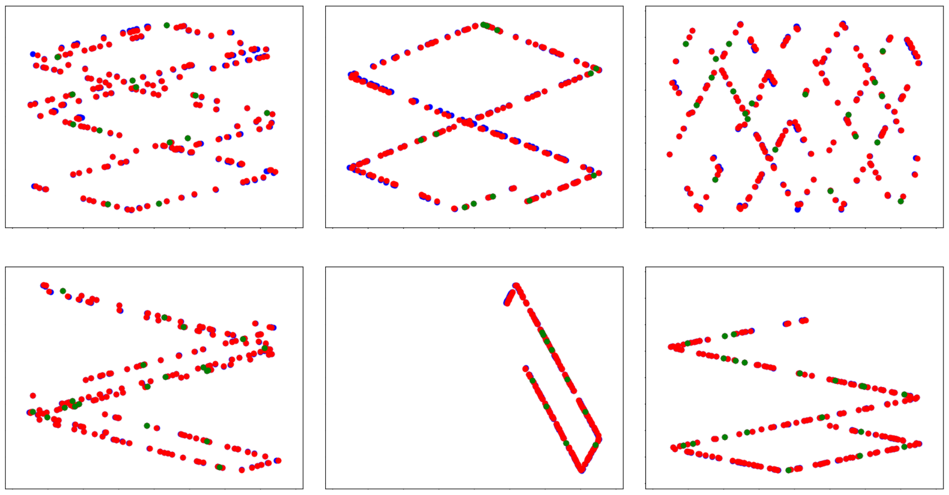


Figure 5.6: Trajectories imputed by NRTSI on the irregularly-sampled Billiards dataset. The red points are the imputed data, the green points are the observed data and the blue points are ground-truth data.

Partially Observed Time Series The air quality dataset [226] and the gas sensor dataset [18] are two popular datasets to evaluate the scenario where feature dimensions at each time point are partially observed. Data in these datasets are 11-dimensional and 19-dimensional respectively. For both datasets, we follow RDIS [33] to select 48 consecutive timesteps to construct one regularly-sampled time series. We compare NRTSI to RDIS, BRITS, Latent-ODE, NeuralCDE, CSDI and mTAN. In Table 5.6, we report the MSE scores by randomly masking out some dimensions for all timesteps with several different missing rates. NRTSI outperforms the baselines on all of the missing rates.

5.7.1 Model Architectures of NRTSI

Our imputation model contains two linear layers f_{in} and f_{out} with bias terms as well as a Transformer encoder f_{enc} that contains N repeated self-attention blocks. We set N to 8 for all the

Table 5.6: The MSE comparison under different missing rates on the air quality dataset and the gas sensor dataset. † results are taken from [33].

Dataset	Method	missing rate			
		20%	40%	60%	80%
Air	Latent-ODE	.2954±.0109	.3291±.0118	.3569±.0124	.3762±.0127
	NeuralCDE	.3129±.0271	.3524±.0285	.4074±.0290	.4865±.0319
	BRITS†	.2076	.2088	.2660	.3421
	RDIS†	.1807	.1977	.2528	.3178
	CSDI	.1236±.0032	.1411±.0041	.1648±.0044	.2155±.0057
	mTAN	.1192±.0034	.1261±.0033	.1403±.0046	.1885±.0049
	NRTSI	**.1155±.0035**	**.1250±.0038**	**.1378±.0039**	**.1790±.0041**
Gas	Latent-ODE	.1282±.0039	.1299±.0041	.1387±.0044	.1979±.0049
	NeuralCDE	.0773±.0024	.1044±.0028	.1538±.0045	.3011±.0097
	BRITS†	.0226	.0279	.0406	.1595
	RDIS†	.0226	.0251	.0321	.0837
	CSDI	.0297±.0009	.0273±.0009	.0352±.0011	.0591±.0017
	mTAN	.0215±.0007	.0259±.0008	.0497±.0012	.0886±.0016
	NRTSI	**.0195±.0007**	**.0229±.0007**	**.0311±.0010**	**.0445±.0012**

datasets reported in our paper except the irregularly-sampled sinusoidal dataset where we set N to 2. Each self-attention block consists of a multi-head self-attention layer followed by a feedforward layer. We use 12 attention heads and every head has a key/query/value pair with a dimension of 128. Then, the output vectors of all heads are concatenated to form a 1,536-dimensional vector and projected to a 1,024-dimensional vector using a linear layer. The feedforward layer contains two linear layers where the first one projects the 1,024-dimensional vector to a 2,048-dimensional vector and the second one projects the 2,048-dimensional vector back to a 1,024-dimensional vector.

Our model with 8 self-attention blocks contains 84.0M learnable parameters.

For all the datasets, we set τ and ν in (5.1) in the main text to 8 and 100 respectively. We set Δ in Section 5.5.3 to 1.

We find that the usually used layer normalization [7] and Dropout [180] techniques in Transformer are detrimental to imputation performance, and they are removed from our model.

In practice, we find it beneficial to add a mask to every attention map to prevent elements in $\mathbf{G}^{(1)}$ from attending to each other. This modification encourages elements in $\mathbf{G}^{(1)}$ to pay more attention to $\mathbf{S}^{(1)}$ and reduces the mutual influence between elements in $\mathbf{G}^{(1)}$.

5.7.2 Training Details of NRTSI

For all the datasets considered in our paper, we use a maximum resolution level of 4, which corresponds to a maximum missing gap of $2^4 = 16$ that our model can impute. Except for the partially observed dimension scenario, every resolution level is taken care of by a corresponding imputation model. For the partially observed dimension scenario, we find a single model is enough. We train NRTSI with Adam [92] using a starting learning rate of 1×10^{-4} and decrease it to 1×10^{-5} when the validation loss reaches a plateau.

When we start training on level l, we adopt the transfer learning strategy to initialize the parameters of f_θ^l from the trained weights of model f_θ^{l+1}. The assumption is that the trained higher-level model already captures the dynamics at a fine-grained scale, and it is beneficial to train lower-level models based on the prior knowledge of higher-level dynamics.

During training, we regard 195 timestamps as missing for both the regularly-sampled and the irregularly-sampled Billiards dataset, 140 timestamps as missing for the traffic dataset, 90 timestamps as missing for the MuJoCo dataset, 40 to 49 timestamps as missing for the football player trajectory dataset, 90 timestamps as missing for the irregularly-sampled sinusoidal dataset. For the gas dataset and the air quality dataset, we use a missing rate of 80% during training.

We use an NVIDIA GeForce GTX 1080 Ti GPU to train our model. We implement our model using PyTorch [148].

5.7.3 Visualization of Attention

Visualizing the attention maps at different self-attention blocks and different heads is helpful to understand how NRTSI imputes missing data.

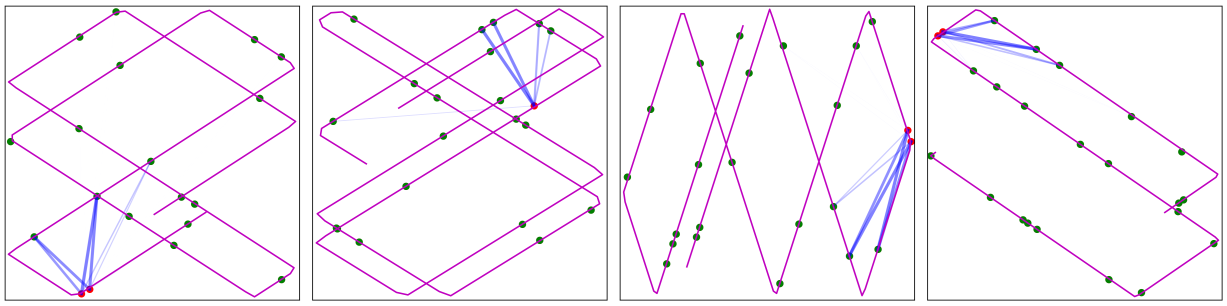


Figure 5.7: Visualization of one attention head. Red points are the imputed data, green points are the observed data. The softmax attention weights are visualized via the blue lines. The wider and less transparent the blue lines are, the larger the attention weights.

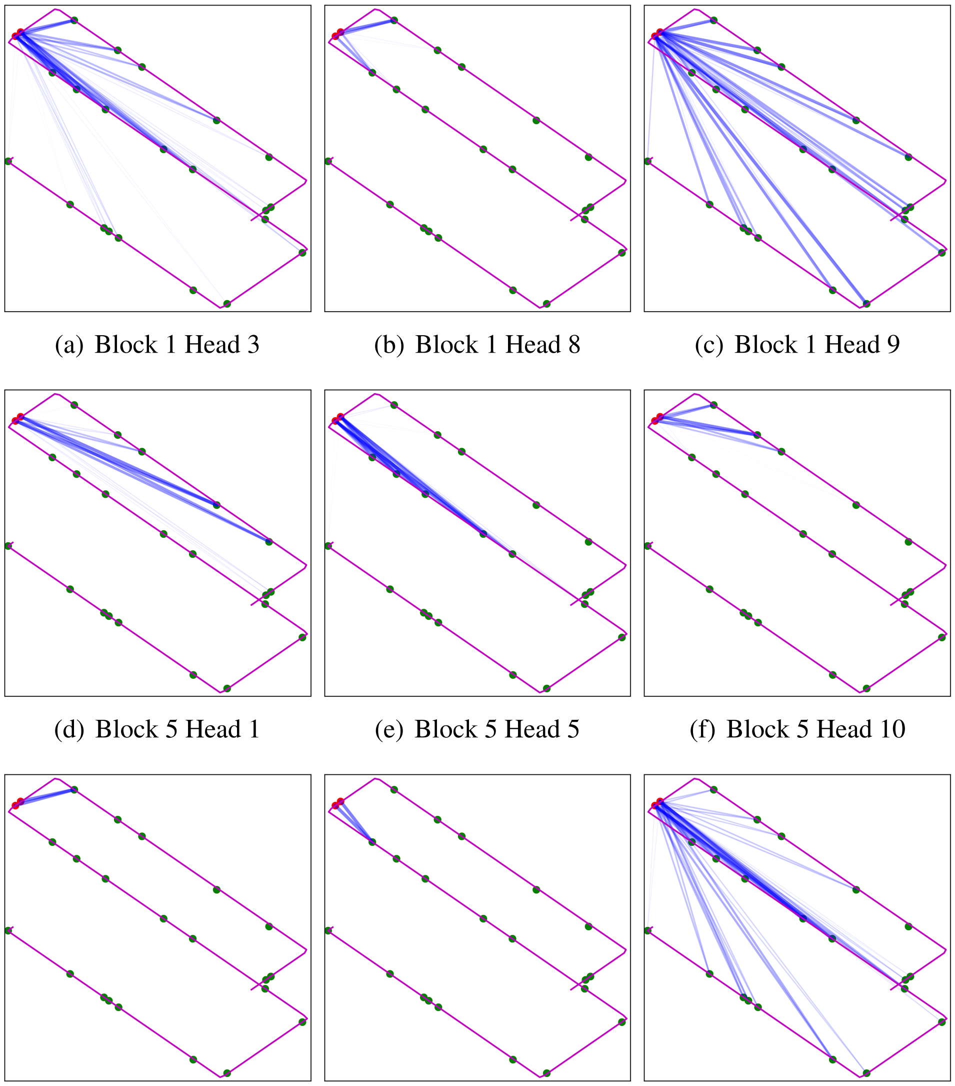
We first visualize the attention maps of an attention head in NRTSI in Fig 5.7 where this attention head models short-range interactions in a single direction. Then, we show several other representative patterns captured by NRTSI in Fig 5.8. According to Fig 5.8, short-range interactions are captured by Block 1 Head 8, Block 5 Head 10, Block 7 Head 5, and Block 7 Head 7. Middle range interactions are captured by Block 1 Head 3, Block 5 Head 1, and Block 5 Head 5. Long-range interactions are captured by Block 1 Head 9 and Block 7 Head 12. It is interesting to find that some attention heads capture interactions on both the forward and backward directions (e.g. Block 1 Head 3, Block 1 head 8, Block 1 Head 9, Block 7 Head 12), while some attention heads capture interactions in a single direction (e.g. Block 5 Head 1, Block 5 Head 5, Block 5 Head 10, Block 7 Head 5, Block 7 Head 7).

5.7.4 Datasets Details

All the datasets can be downloaded or generated from the codes submitted. In the "readme.md" file, links to download these datasets can be found. We also include the python scripts to generate the irregularly-sampled Billiards dataset and the irregularly-sampled sinusoidal function dataset.

5.7.4.1 Billiards Dataset

Billiards dataset [164] contains 4,000 training and 1,000 testing sequences, which are regularly-sampled trajectories of Billiards balls in a rectangular world. We download this dataset from the


(a) Block 1 Head 3 (b) Block 1 Head 8 (c) Block 1 Head 9

(d) Block 5 Head 1 (e) Block 5 Head 5 (f) Block 5 Head 10

(g) Block 7 Head 5 (h) Block 7 Head 7 (i) Block 7 Head 12

Figure 5.8: Visualization of the attention maps at different self-attention blocks and different heads on the regularly-sampled Billiard dataset. There are 8 self-attention blocks with 12 heads each in our model. Red points are the imputed data, green points are the observed data and the purple solid lines are ground-truth trajectories. The softmax attention weights are visualized via the blue lines. The wider and less transparent the blue lines are, the larger the attention weights.

official GitHub repository [1] of NAOMI [123] to guarantee our results are comparable to the results in the NAOMI paper. Note that this dataset does not have a validation set because NAOMI does not have a validation set. To make our results comparable to NAOMI we have to follow its setup.

5.7.4.2 PEMS-SF traffic dataset

The PEMS-SF traffic [43] dataset contains 267 training and 173 testing sequences of length 144 (regularly-sampled every 10 mins throughout the day). We follow NAOMI to download this dataset form `https://archive.ics.uci.edu/ml/datasets/PEMS-SF` and follow NAOMI to use the same default training/testing split. Note that this dataset does not have a validation set because NAOMI does not have a validation set. To make our results comparable to NAOMI we have to follow its setup.

5.7.4.3 MuJoCo dataset

The dataset is 14-dimensional and contains 10,000 sequences of 100 regularly-sampled time points for each trajectory. We download this dataset from the official GitHub repository [2] of Latent ODE [162] to guarantee our results are comparable to the results in Latent ODE paper. We follow Latent ODE to randomly split the dataset into 80% train and 20% test with the random seed. Note that this dataset does not have a validation set because Latent ODE does not have a validation set. To make our results comparable to Latent ODE we have to follow its setup.

5.7.4.4 Irregularly-sampled Billiards Dataset

This dataset is created with the same parameters (e.g. initial ball speed range, travel time) to create its regularly-sampled counterpart in Section 4.1 in the main text. The only difference is that this dataset is irregularly-sampled. The billiard table is square and the side length of the

[1] `https://github.com/felixykliu/NAOMI`

[2] `https://github.com/YuliaRubanova/latent_ode`

square is 0.8828. The initial ball speed range is 0.0018 to 0.1075. The travel time for each trajectory is 200 seconds. We randomly generate 12,000 trajectories for training and 1,000 trajectories for testing.

5.7.4.5 The Gas Sensor Dataset and the Air Quality Dataset

We strictly follow RDIS [33] to use the same strategy to prepare the gas sensor dataset and the air quality dataset. Please refer to the RDIS paper for details. For a fair comparison, we use the same random seed to perform train/validation/test split as done in RDIS.

5.7.4.6 Football Player Trajectory Dataset

This dataset contains 9,543 time series with 50 regularly-sampled time points for each trajectory. The sampling rate is fixed at 10 Hz. We collect this football player trajectory dataset from the 2021 Big Data Bowl data [86]. For American football games, the number of players on each team is 11. However, data of linemen are not provided in [86]. As a result, the number of players is less than 11. To collect a dataset with the same number of players, we only use data with exactly 6 offensive players. 8,000 trajectories are randomly selected for training and the remaining ones are used for testing.

This dataset is publicly available and does not contain personally identifiable information or offensive content.

5.7.5 Ablation Studies of NRTSI

5.7.5.1 Ablation Studies of the Essential Components

In this section, we perform several ablation studies on the regularly-sampled Billiard dataset to evaluate the contribution of each component in NRTSI. We consider four variants of NRTSI and report their L2 losses in Table 5.7.

Table 5.7: Ablation studies of NRTSI. All the methods use the same Transformer encoder architecture hyperparameters according to Section 5.7.1.

Method Name	Description	L2 loss ($\times 10^{-2}$)
`NRTSI w/o Individual`	A variant of NRTSI that uses a single imputation model to impute at all hierarchies.	0.16
`NRTSI w/o Encode` $\mathbf{S} \cup \mathbf{G}$	A variant of NRTSI that uses the Transformer encoder to only model $\mathbf{S}$. Then $\mathbf{G}$ attends to the representation of $\mathbf{S}$ via cross-attention.	0.27
`NRTSI w/o Hierarchy + all`	A variant of NRTSI that imputes all missing data at once without the proposed hierarchical strategy.	1.14
`NRTSI w/o Hierarchy + small`	A variant of NRTSI that imputes data with the smallest missing gap first without the proposed hierarchical strategy.	11.09
NRTSI	The proposed method.	**0.024**

From Table 5.7, it can be seen that not using the proposed hierarchical imputation strategy (`NRTSI w/o Hierarchy`) deteriorates the performance most. More specifically, the variant that imputes data with the smallest missing gap first (`NRTSI w/o Hierarchy + small`) is very similar to recurrent methods that always impute data at the next nearby timestamps first. As a result, this variant suffers from the error compounding problem as shown qualitatively in Fig 5.9. Another variant that imputes all the missing data at once (`NRTSI w/o Hierarchy + all`) is similar to the way that ANP [90] imputes. We show its performance quantitatively in Table 5.7 and qualitatively in Fig 5.10.

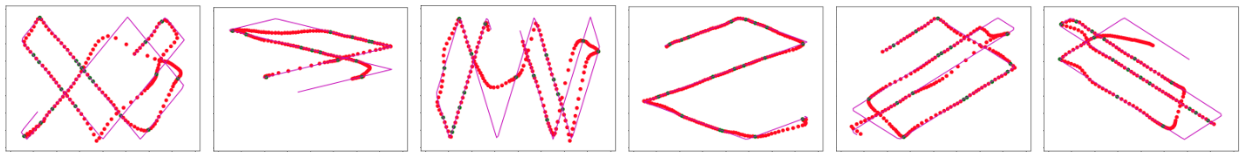


Figure 5.9: Qualitative results of `NRTSI w/o Hierarchy + small` on the Billiards dataset. The purple solid lines are ground-truth trajectories, the red points are imputed data and the green points are observed data.

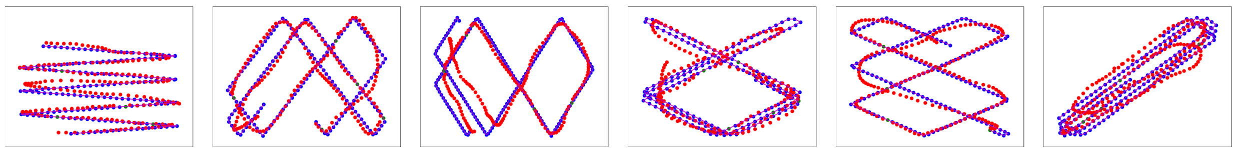


Figure 5.10: Qualitative results of `NRTSI w/o Hierarchy + all` on the Billiards dataset. The purple solid lines and the blue points are ground-truth trajectories, the red points are imputed data and the green points are observed data.

Another variant `NRTSI w/o Encode` $\mathbf{S} \cup \mathbf{G}$ only uses the Transformer encoder to model $\mathbf{S}$ rather than $\mathbf{S} \cup \mathbf{G}$. Then $\mathbf{G}$ attends to the representation of $\mathbf{S}$ computed by the Transformer encoder via the cross-attention mechanism. This variant also deteriorates the performance of NRTSI because the information of what timesteps to impute (i.e. $\mathbf{G}$) is not utilized when the Transformer encoder uses self-attention to compute the representations of observed data (i.e. $\mathbf{S}$). Therefore, the representation might be suboptimal. This model architecture is very similar to ANP [90] that also ignores the target input when ANP uses self-attention to compute the representations of the context input/output pairs. This shortcoming and the fact that ANP imputes all missing data at once together explain why NRTSI outperforms ANP by a large margin in Table 5.5.

Using a single model to impute at all hierarchy levels (`NRTSI w/o Individual`) also slightly deteriorates the performance. This is because imputing at different hierarchical levels requires different types of attention patterns, and it might be hard to capture all the necessary patterns in a single model.

5.7.6 Discussion

Throughout the experiments, we conduct an extensive hyperparameter searching for the baselines to make sure they perform as well as they can. We find that the best configurations of these baselines are not improved by increasing their model capacities. Thus, the superiority of NRTSI is due to the novel architecture rather than naively using more parameters. In terms of the imputation speed, though NRTSI contains more parameters than NAOMI, the increase of the number of parameters does not slow down the imputation speed of NRTSI. According to Table 5.4, NRTSI is faster than NAOMI as NRTSI can impute multiple missing data in parallel. This speedup is also observed on other datasets. To evaluate the contribution of each component in NRTSI, we conduct several ablation studies in Section 5.7.5.1 and find that both the hierarchical imputation procedure and the proposed attention mechanism are required to achieve good performance.

5.8 Closing Remarks

In this work, we introduce a novel time-series imputation approach named NRTSI. NRTSI represents time series as permutation equivariant sets and leverages a Transformer-based architecture to impute the missing values. We also propose a hierarchical imputation procedure where missing data are imputed in the order of their missing gaps (i.e. the distance to the closest observations). NRTSI is broadly applicable to numerous applications, such as irregularly-sampled time series, partially observed time series, and stochastic time series. Extensive experiments demonstrate that NRTSI achieves state-of-the-art performance on commonly used time series imputation benchmarks.

Regarding limitations, the optimal number of hierarchy levels needs to be tuned individually for different datasets. Another limitation is that for stochastic imputation, we trade off some degree of parallelism for trajectory quality. In the future, we hope to introduce a global latent variable to control the overall stochasticity of imputed trajectories, rather than stochastically imputing every single missing data individually.

CHAPTER 6: CONCLUSION

Data is one of the most important parts of machine learning. We need data to train machine learning models and use the trained models to make predictions on new data. However, most existing models consider every individual data instance in isolation. In this dissertation, we show the benefits of also learning and exploiting related data instances to enhance performance. To better leverage related instances, several novel models equipped with dictionary learning, attention mechanisms, and kernel methods are proposed. Extensive experiments on different types of data are conducted to support our proposal. We hope our findings can shed some light on future research that can better model the relationship between data instances to further improve machine learning performance.

6.1 Review of Thesis Aims

6.1.1 Leveraging Related Neighbor Instances

First, we consider the case where the input is a single instance. To improve prediction performance, we equip neural networks with a dictionary of learnable neighbor instances. During training, these related instances are automatically learned in a data-driven fashion. During inference, the related instances for a specific input instance are retrieved and leveraged for better prediction. We show the benefit of exploiting related neighbor instances to obtain an input-specific predictive model with higher accuracies and lower regression errors. We later extend this idea to learn related waveform instances for efficient and fast audio synthesis. For both studies, we find our learned related instances are helpful to improve model performance and also semantically meaningful to provide some understanding of how the decision is made by our models. This prediction paradigm leveraging related instances is also similar to the human reasoning process, as human beings frequently pull similar experiences from memory to assist decision-making. Thus, our proposed approaches resemble the human reasoning strategy and may inspire future works in this direction.

6.1.2 Leveraging Related Points from a Collection

We also consider the case where the input is a collection of instances, such as point clouds, time series, and protein expressions across a set of cells. In these cases, we are directly given a collection of related instances and we need to consider them in aggregation to make a prediction over the collection holistically. When the collection is a set, we develop specifically designed models to respect the unordered property of sets. We first develop a kernel-based method for the single-cell sample sets classification task and our model can provide rich biological interpretability. We also propose another method for time series imputation where time series are viewed as sets and a hierarchical imputation procedure is used to improve performance. Our findings suggest that it is beneficial to develop models that can comprehensively reason about the relationship between related instances belonging to the same collection.

6.2 Future Works

6.2.1 Leveraging Related Instances across Modalities

In this dissertation, we only leverage related instances from the same modality. For instance, to improve image classification performance, we only consider neighbors that are also images; when reconstructing waveforms we only consider neighbors that are also waveforms; to predict the clinical outcomes of single-cell sample sets, all the related instances are protein expression features. However, in the real world, not all related instances belong to the same modality. For instance, an image may have several related audio instances that describe the content of the image [55, 202]; a short record of speech audio may have several related text transcripts; a synthetic image may have a corresponding real image [125]; Thus, one future direction is to also leverage related instances across modalities. For instance, a recent work CLIP [152] shows that a more robust feature space can be learned thanks to visual/text joint learning where the training data are images with related text descriptions. CLIP leverages related text instances of images rather than only considering images alone and thus can extract more robust features. Several following works inspired by CLIP also achieve significant advancements like text-conditioned image generation and visual-conditioned text generation [4, 156]. Future works could focus on leveraging related instances for representation learning and generation across other different modalities, such as caption-image-sound joint learning or text-guided music generation. To achieve this goal, we need to collect datasets containing paired instances from different modalities. To reduce the human efforts in collecting data, we may resort to web crawler techniques to automatically discover paired data from the Internet.

6.2.2 Leveraging Related Instances at Scale

Another future direction is to learn related instances at a larger scale. In our work Meta-Neighborhood, the learnable dictionary only contains 5,000 to 10,000 related instances. However, recent works such as CLIP [152] and ChatGPT [1] [83, 216] show that the model performance can be significantly improved by training on millions of related instances. Therefore, we need to develop models that are more expressive to fit increasing amounts of training data. A more expressive and larger model is also often harder to optimize and we need to develop better optimization, initialization, and regularization methods [16, 219, 142, 73, 109, 108]. We also need to develop more capable computation infrastructures so that large-scale training can be finished within a reasonable amount of time.

When the input is a collection of instances, we also need to handle cases where the cardinality of the collection is huge. We show that our work CKME can efficiently handle a collection with millions of elements with kernel herding. However, kernel herding is a greedy approach to sub-select the collection and it is more desirable to select in a data-driven fashion, such as us-

[1]`https://chat.openai.com/`

ing attention mechanisms. However, the original self-attention mechanism scales poorly to long sequences. Thus, we need to develop new attention mechanisms that scale well to huge collections. Several recent works such as Linformer [203] and Reformer [94] have reduced the time complexity of the self-attention mechanism from $O(n^2)$ to $O(n)$ and $O(nlog(n))$ respectively using low-rank approximation or locally sensitive hashing. Future works can build new attention mechanisms that scale well based on the property of data.

REFERENCES

[1] `https://flossmanual.csound.com/sound-synthesis/additive-synthesis`. [Online; accessed 10-Jan-2023].

[2] Edward H Adelson, Charles H Anderson, James R Bergen, Peter J Burt, and Joan M Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

[3] Nima Aghaeepour, Greg Finak, Holger Hoos, Tim R Mosmann, Ryan Brinkman, Raphael Gottardo, and Richard H Scheuermann. Critical assessment of automated flow cytometry data analysis techniques. *Nature methods*, 10(3):228–238, 2013.

[4] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[5] Eirini Arvaniti and Manfred Claassen. Sensitive detection of rare disease-associated cell subsets via representation learning. *Nature communications*, 8(1):1–10, 2017.

[6] Kavosh Asadi, Dipendra Misra, and Michael Littman. Lipschitz continuity in model-based reinforcement learning. In *International Conference on Machine Learning*, pages 264–273. PMLR, 2018.

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] Vishal Athreya Baskaran, Jolene Ranek, Siyuan Shan, Natalie Stanley, and Junier B. Oliva. Distribution-based sketching of single-cell samples. In *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '22, New York, NY, USA, 2022. Association for Computing Machinery.

[10] James W Beauchamp. *Analysis, synthesis, and perception of musical sounds*. Springer, 2007.

[11] Sean C Bendall et al. Single-cell mass cytometry of differential immune and drug responses across a human hematopoietic continuum. *Science*, 332(6030):687–696, 2011.

[12] Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

[13] Julian D. Parker Boris Kuznetsov and Fabián Esqueda. Differentiable iir filters for machine learning applications. *DaFX*, 2020.

[14] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and Music Signal Analysis in Python. In *Proceedings of the 14th Python in Science Conference*, pages 18 – 24, 2015.

[15] Robert Bristow-Johnson. Wavetable synthesis 101, a fundamental perspective. In *Audio Engineering Society Convention 101*. Audio Engineering Society, 1996.

[16] Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*, pages 1059–1071. PMLR, 2021.

[17] Robert V Bruggner, Bernd Bodenmiller, David L Dill, Robert J Tibshirani, and Garry P Nolan. Automated identification of stratifying signatures in cellular subpopulations. *Proceedings of the National Academy of Sciences*, 111(26):E2770–E2777, 2014.

[18] Javier Burgués, Juan Manuel Jiménez-Soto, and Santiago Marco. Estimation of the limit of detection in semiconductor gas sensors through linearized calibration models. *Analytica chimica acta*, 1013:13–25, 2018.

[19] Daniel B Burkhardt, Jay S Stanley, Alexander Tong, Ana Luisa Perdigoto, Scott A Gigante, Kevan C Herold, Guy Wolf, Antonio J Giraldez, David van Dijk, and Smita Krishnaswamy. Quantifying the effect of experimental perturbations at single-cell resolution. *Nature Biotechnology*, pages 1–11, 2021.

[20] Qi Cai, Yingwei Pan, Ting Yao, Chenggang Yan, and Tao Mei. Memory matching networks for one-shot image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4080–4088, 2018.

[21] Estefanía Cano. *Pitch-informed solo and accompaniment separation*. PhD thesis, 10 2014.

[22] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. Brits: Bidirectional recurrent imputation for time series. In *Advances in Neural Information Processing Systems*, pages 6775–6785, 2018.

[23] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.

[24] Tinnakorn Chaiworapongsa, Maria-Teresa Gervasi, Jerrie Refuerzo, Jimmy Espinoza, Jun Yoshimatsu, Susan Berman, and Roberto Romero. Maternal lymphocyte subpopulations (CD45RA+ and CD45RO+) in preeclampsia. *Am. J. Obstet. Gynecol.*, 187(4):889–893, oct 2002.

[25] Cheng Chang, Runzhe Yang, Lu Chen, Xiang Zhou, and Kai Yu. Affordable on-line dialogue policy learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2200–2209, 2017.

[26] F. Charpentier and M. Stella. Diphone synthesis using an overlap-add technique for speech waveforms concatenation. In *ICASSP*, volume 11, pages 2015–2018, 1986.

[27] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.

[28] Lu Chen, Xiang Zhou, Cheng Chang, Runzhe Yang, and Kai Yu. Agent-aware dropout dqn for safe and efficient on-line dialogue policy learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2454–2464, 2017.

[29] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[30] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *International Conference on Learning Representations*, 2019.

[31] Yutian Chen and Max Welling. Parametric herding. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 97–104, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[32] Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pages 109–116, 2010.

[33] Tae-Min Choi, Ji-Su Kang, and Jong-Hwan Kim. Rdis: Random drop imputation with self-training for incomplete time series data. *AAAI*, 2021.

[34] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[35] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[36] Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

[37] D Darmochwal-Kolarz, S Saito, J Rolinski, J Tabarkiewicz, B Kolarz, B Leszczynska-Gorzelak, and J Oleszczuk. Activated T lymphocytes in pre-eclampsia. *Am. J. Reprod. Immunol.*, 58(1):39–45, jul 2007.

[38] Mark M Davis, Cristina M Tato, and David Furman. Systems immunology: just getting started. *Nature immunology*, 18(7):725, 2017.

[39] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. In *Advances in Neural Information Processing Systems*, pages 7379–7390, 2019.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[42] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[43] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[44] Jesse Engel, Chenjie Gu, Adam Roberts, et al. Ddsp: Differentiable digital signal processing. In *International Conference on Learning Representations*, 2019.

[45] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. Ddsp: Differentiable digital signal processing. *ICLR*, 2020.

[46] Jesse Engel, Lamtharn Hantrakul, Rigel Swavely, Adam Roberts, and Curtis Glenn-Macway Hawthorne. Self-supervised pitch detection by inverse audio synthesis. In *ICML SAS Workshop*, 2020.

[47] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. In *ICML*, 2017.

[48] Jesse H. Engel, Kumar Krishna Agrawal, Shuo Chen, I. Gulrajani, C. Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *ICLR*, 2019.

[49] Ramin Fallahzadeh, Franck Verdonk, et al. Objective activity parameters track patient-specific physical recovery trajectories after surgery and link with individual preoperative immune states. *Annals of Surgery*, 2021.

[50] Zhenghan Fang, Yong Chen, Mingxia Liu, Lei Xiang, Qian Zhang, Qian Wang, Weili Lin, and Dinggang Shen. Deep learning for fast and spatially constrained tissue quantification from highly accelerated data in magnetic resonance fingerprinting. *IEEE transactions on medical imaging*, 38(10):2364–2374, 2019.

[51] William Fedus, Ian Goodfellow, and Andrew M Dai. Maskgan: Better text generation via filling in the_. *ICLR*, 2018.

[52] Zishun Feng, Dong Nie, Li Wang, and Dinggang Shen. Semi-supervised learning for pelvic mr image segmentation based on multi-task residual fully convolutional networks. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 885–888. IEEE, 2018.

[53] Zishun Feng, Joseph A. Sivak, and Ashok K. Krishnamurthy. Two-stream attention spatio-temporal network for classification of echocardiography videos. In *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, pages 1461–1465, 2021.

[54] Zishun Feng, Joseph A. Sivak, and Ashok K. Krishnamurthy. Improving echocardiography segmentation by polar transformation. In Oscar Camara, Esther Puyol-Antón, Chen Qin, Maxime Sermesant, Avan Suinesiaputra, Shuo Wang, and Alistair Young, editors, *Statistical Atlases and Computational Models of the Heart. Regular and CMRxMotion Challenge Papers*, pages 133–142, Cham, 2022. Springer Nature Switzerland.

[55] Zishun Feng, Ming Tu, Rui xia, Yuxuan Wang, and Ashok Krishnamurthy. Self-supervised audio-visual representation learning for in-the-wild videos. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 5671–5672, 2020.

[56] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[57] Simon Foucart and Jiangyuan Li. Sparse recovery from inaccurate saturated measurements. *Acta Applicandae Mathematicae*, 158(1):49–66, 2018.

[58] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[59] Edward A Ganio, Natalie Stanley, Viktoria Lindberg-Larsen, et al. Preferential inhibition of adaptive immune system dynamics by glucocorticoids in patients after acute surgical trauma. *Nature communications*, 11(1):1–12, 2020.

[60] Francesco Ganis, Erik Frej Knudesn, Søren VK Lyster, Robin Otterbein, David Südholt, and Cumhur Erkut. Real-time timbre transfer and sound synthesis using ddsp. *arXiv preprint arXiv:2103.07220*, 2021.

[61] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR, 2018.

[62] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

[63] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, 2018.

[64] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[65] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv:1410.5401*, 2014.

[66] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

[67] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel method for the two-sample problem. *Journal of Machine Learning Research*, 1:1–10, 2008.

[68] Xiaoyuan Han, Mohammad S Ghaemi, et al. Differential dynamics of the maternal immune system in healthy pregnancy and preeclampsia. *Frontiers in immunology*, 10:1305, 2019.

[69] Lamtharn Hantrakul, Jesse H Engel, Adam Roberts, and Chenjie Gu. Fast and flexible neural audio synthesis. In *ISMIR*, 2019.

[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[71] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[72] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[73] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[74] Max Horn, Michael Moor, Christian Bock, Bastian Rieck, and Karsten Borgwardt. Set functions for time series. In *International Conference on Machine Learning*, pages 4353–4363. PMLR, 2020.

[75] Andrew Horner. Wavetable matching synthesis of dynamic instruments with genetic algorithms. 43:916–931, 11 1995.

[76] Andrew Horner, James Beauchamp, and Lippold Haken. Methods for multiple wavetable synthesis of musical instrument tones. *Journal of the Audio Engineering Society*, 41(5):336–356, 1993.

[77] Andrew B. Horner and James W. Beauchamp. *Instrument Modeling and Synthesis*, pages 375–397. Springer New York, New York, NY, 2008.

[78] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[79] Zicheng Hu, Benjamin S Glicksberg, and Atul J Butte. Robust prediction of clinical outcomes using cytometry data. *Bioinformatics*, 35(7):1197–1203, 2019.

[80] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[81] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[82] Bahsima Islam, Yubo Luo, Seulki Lee, and Shahriar Nirjon. On-device training from sensor data on batteryless platforms. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, pages 325–326, 2019.

[83] Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Xing Wang, and Zhaopeng Tu. Is chatgpt a good translator? a preliminary study. *arXiv preprint arXiv:2301.08745*, 2023.

[84] Xin Jin, Sunil Manandhar, Kaushal Kafle, Zhiqiang Lin, and Adwait Nadkarni. Understanding iot security from a market-scale perspective. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1615–1629, 2022.

[85] Xin Jin, Kexin Pei, Jun Yeon Won, and Zhiqiang Lin. Symlm: Predicting function names in stripped binaries via context-sensitive execution-aware code embeddings. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1631–1645, 2022.

[86] Kaggle. Nfl big data bowl 2021. `https://www.kaggle.com/c/nfl-big-data-bowl-2021`, 2020. Accessed: 2021-01-02.

[87] Łukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to remember rare events. *arXiv preprint arXiv:1703.03129*, 2017.

[88] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *ICLR*, 2018.

[89] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *NeurIPS*, 2020.

[90] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2018.

[91] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. Crepe: A convolutional representation for pitch estimation. In *ICASSP*, 2018.

[92] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[93] Vladimir Yu Kiselev, Tallulah S Andrews, and Martin Hemberg. Challenges in unsupervised clustering of single-cell RNA-seq data. *Nat. Rev. Genet.*, 20(5):273–282, may 2019.

[94] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[95] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[96] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[97] Patricia K Kuhl, Feng-Ming Tsao, and Huei-Mei Liu. Foreign-language experience in infancy: Effects of short-term exposure and social interaction on phonetic learning. *Proceedings of the National Academy of Sciences*, 100(15):9096–9101, 2003.

[98] Zhengfeng Lai, Chao Wang, Henrry Gunawan, Sen-Ching S Cheung, and Chen-Nee Chuah. Smoothed adaptive weighting for imbalanced semi-supervised learning: Improve reliability against unknown distribution data. In *International Conference on Machine Learning*, pages 11828–11843. PMLR, 2022.

[99] Zhengfeng Lai, Chao Wang, Luca Cerny Oliveira, Brittany N Dugger, Sen-Ching Cheung, and Chen-Nee Chuah. Joint semi-supervised and active learning for segmentation of gigapixel pathology images with cost-effective labeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 591–600, 2021.

[100] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series.

[101] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[102] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019.

[103] Seulki Lee, Bashima Islam, Yubo Luo, and Shahriar Nirjon. Intermittent learning: On-device machine learning on intermittently powered system. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3(4):1–30, 2019.

[104] Sungho Lee, Hyeong-Seok Choi, and Kyogu Lee. Differentiable artificial reverberation. *CoRR*, abs/2105.13940, 2021.

[105] Bochen Li, Xinzhao Liu, K. Dinesh, Z. Duan, and Gaurav Sharma. Creating a multitrack classical music performance dataset for multimodal music analysis: Challenges, insights, and applications. *IEEE Transactions on Multimedia*, 21(2):522–535, 2018.

[106] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550, 2017.

[107] Jiangyuan Li and Mohammadreza Armandpour. Deep spatio-temporal wind power forecasting. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4138–4142, 2022.

[108] Jiangyuan Li, Thanh Nguyen, Chinmay Hegde, and Ka Wai Wong. Implicit sparse regularization: The impact of depth and early stopping. *Advances in Neural Information Processing Systems*, 34:28298–28309, 2021.

[109] Jiangyuan Li, Thanh V Nguyen, Chinmay Hegde, and Raymond K. W. Wong. Implicit regularization for group sparsity. In *The Eleventh International Conference on Learning Representations*, 2023.

[110] Xiaomin Li and Vangelis Metsis. Spp-eegnet: An input-agnostic self-supervised eeg representation model for inter-dataset transfer learning. In Phayung Meesad, Sunantha Sodsee, Watchareewan Jitsakul, and Sakchai Tangwannawit, editors, *Proceedings of the 18th International Conference on Computing and Information Technology (IC2IT 2022)*, pages 173–182, Cham, 2022. Springer International Publishing.

[111] Xiaomin Li, Vangelis Metsis, Huangyingrui Wang, and Anne Hee Hiong Ngu. Tts-gan: A transformer-based time-series generative adversarial network. In *Artificial Intelligence in Medicine: 20th International Conference on Artificial Intelligence in Medicine, AIME 2022, Halifax, NS, Canada, June 14–17, 2022, Proceedings*, pages 133–143. Springer, 2022.

[112] Xiaomin Li, Anne Hee Hiong Ngu, and Vangelis Metsis. Tts-cgan: A transformer time-series conditional gan for biosignal data augmentation. *arXiv preprint arXiv:2206.13676*, 2022.

[113] Yang Li, Siyuan Shan, Qin Liu, and Junier B Oliva. Towards robust active feature acquisition. *arXiv preprint arXiv:2107.04163*, 2021.

[114] Yang Li, Haidong Yi, Christopher Bender, Siyuan Shan, and Junier B Oliva. Exchangeable neural ode for set modeling. *Advances in Neural Information Processing Systems*, 33:6936–6946, 2020.

[115] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[116] Ziyan Li, Jianjiang Feng, Zishun Feng, Yunqiang An, Yang Gao, Bin Lu, and Jie Zhou. Lumen segmentation of aortic dissection with cascaded convolutional network. In *Statistical Atlases and Computational Models of the Heart. Atrial Segmentation and LV Quantification Challenges: 9th International Workshop, STACOM 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16, 2018, Revised Selected Papers 9*, pages 122–130. Springer, 2019.

[117] Thomas Liechti, Lukas M Weber, Thomas M Ashhurst, Natalie Stanley, Martin Prlic, Sofie Van Gassen, and Florian Mair. An updated guide for the perplexed: cytometry in the high-dimensional era. *Nature Immunology*, 22(10):1190–1197, 2021.

[118] Honghui Liu, Jianjiang Feng, Zishun Feng, Jiwen Lu, and Jie Zhou. Left atrium segmentation in ct volumes with fully convolutional networks. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*, pages 39–46. Springer, 2017.

[119] Jiali Liu, Wenxuan Wang, Tianyao Guan, Ningbo Zhao, Xiaoguang Han, and Zhen Li. Ultrasound liver fibrosis diagnosis using multi-indicator guided deep neural networks. In *Machine Learning in Medical Imaging: 10th International Workshop, MLMI 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 13, 2019, Proceedings 10*, pages 230–237. Springer, 2019.

[120] Meijun Liu, Jicong Zhang, Wenxiao Jia, Qi Chang, Siyuan Shan, Yegang Hu, and Dangxiao Wang. Enhanced executive attention efficiency after adaptive force control training: Behavioural and physiological results. *Behavioural Brain Research*, 376:111859, 2019.

[121] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. In *International Conference on Learning Representations*, 2018.

[122] Shikun Liu, Andrew Davison, and Edward Johns. Self-supervised generalisation with meta auxiliary learning. In *Advances in Neural Information Processing Systems*, pages 1677–1687, 2019.

[123] Yukai Liu, Rose Yu, Stephan Zheng, Eric Zhan, and Yisong Yue. Naomi: Non-autoregressive multiresolution sequence imputation. In *Advances in Neural Information Processing Systems*, pages 11238–11248, 2019.

[124] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[125] Conny Lu, Qian Zhang, Kapil Krishnakumar, Jixu Chen, Henry Fuchs, Sachin Talathi, and Kun Liu. Geometry-aware eye image-to-image translation. In *2022 Symposium on Eye Tracking Research and Applications*, pages 1–7, 2022.

[126] Yonghong Luo, Xiangrui Cai, Ying Zhang, Jun Xu, et al. Multivariate time series imputation with generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 1596–1607, 2018.

[127] Yonghong Luo, Xiangrui Cai, Ying ZHANG, Jun Xu, and Yuan xiaojie. Multivariate time series imputation with generative adversarial networks. In *Advances in Neural Information Processing Systems 31*, pages 1596–1607. Curran Associates, Inc., 2018.

[128] Yubo Luo and Yongfeng Huang. Text steganography with high embedding rate: Using recurrent neural networks to generate chinese classic poetry. In *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia Security*, IHMMSec '17, page 99–104, New York, NY, USA, 2017. Association for Computing Machinery.

[129] Yubo Luo, Yongfeng Huang, Fufang Li, and Chinchen Chang. Text steganography based on ci-poetry generation using markov chain model. *KSII Transactions on Internet and Information Systems (TIIS)*, 10(9):4568–4584, 2016.

[130] Yubo Luo and Shahriar Nirjon. Spoton: Just-in-time active event detection on energy autonomous sensing systems. *Brief Presentations Proceedings (RTAS 2019)*, 9, 2019.

[131] Yubo Luo and Shahriar Nirjon. Smarton: Just-in-time active event detection on energy harvesting systems. In *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 35–44. IEEE, 2021.

[132] Yubo Luo, Le Zhang, Zhenyu Wang, and Shahriar Nirjon. Efficient multitask learning on resource-constrained systems. *arXiv preprint arXiv:2302.13155*, 2023.

[133] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[134] Jiawei Ma, Zheng Shou, Alireza Zareian, Hassan Mansour, Anthony Vetro, and Shih-Fu Chang. Cdsa: Cross-dimensional self-attention for multivariate, geo-tagged time series imputation. *arXiv preprint arXiv:1905.09904*, 2019.

[135] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[136] R. McAulay and T. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4):744–754, 1986.

[137] Sebastian Mika, Bernhard Schölkopf, Alexander J Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel pca and de-noising in feature spaces. In *NIPS*, volume 11, pages 536–542, 1998.

[138] Tomas Mikolov, I. Sutskever, K. Chen, Greg S Corrado, and J Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[139] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[140] Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, and Bernhard Schölkopf. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends in Machine Learning*, 10(1-2):1–141, 2017.

[141] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2554–2563. JMLR. org, 2017.

[142] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

[143] Junier Oliva, Willie Neiswanger, Barnabás Póczos, Jeff Schneider, and Eric Xing. Fast distribution to real regression. In *Artificial Intelligence and Statistics*, pages 706–714. PMLR, 2014.

[144] Junier B Oliva, Danica J Sutherland, Barnabás Póczos, and Jeff Schneider. Deep mean maps. *arXiv preprint arXiv:1511.04150*, 2015.

[145] Luca Cerny Oliveira, Zhengfeng Lai, Heather M Siefkes, and Chen-Nee Chuah. Generalizable semi-supervised learning strategies for multiple learning tasks using 1-d biomedical signals. In *NeurIPS 2022 Workshop on Learning from Time Series for Health*, 2022.

[146] Eunbyung Park and Junier B Oliva. Meta-curvature. In *Advances in Neural Information Processing Systems 32*, pages 3309–3319. 2019.

[147] Seong Hyeon Park, Gyubok Lee, Manoj Bhat, Jimin Seo, Minseok Kang, Jonathan Francis, Ashwin R Jadhav, Paul Pu Liang, and Louis-Philippe Morency. Diverse and admissible trajectory forecasting through multimodal context understanding. *ECCV*, 2020.

[148] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[149] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[150] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[151] Peng Qiu, Erin F Simonds, Sean C Bendall, Kenneth D Gibbs, Robert V Bruggner, Michael D Linderman, Karen Sachs, Garry P Nolan, and Sylvia K Plevritis. Extracting a cellular hierarchy from high-dimensional cytometry data with spade. *Nature biotechnology*, 29(10):886–891, 2011.

[152] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 18–24 Jul 2021.

[153] Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *NIPS*, volume 3, page 5. Citeseer, 2007.

[154] Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew M Dai, Nissan Hajaj, Michaela Hardt, Peter J Liu, Xiaobing Liu, Jake Marcus, Mimi Sun, et al. Scalable and accurate deep learning with electronic health records. *NPJ Digital Medicine*, 1(1):18, 2018.

[155] Tiago Ramalho and Marta Garnelo. Adaptive posterior learning: few-shot learning with a surprise-based memory module. In *International Conference on Learning Representations*, 2019.

[156] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[157] Marco A Martínez Ramírez, Oliver Wang, Paris Smaragdis, and Nicholas J Bryan. Differentiable signal processing with black-box audio effects. In *ICASSP*, pages 66–70, 2021.

[158] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ”why should I trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[159] Alexander Richard, Dejan Markovic, Israel D Gebru, Steven Krenn, Gladstone Alexander Butler, Fernando Torre, and Yaser Sheikh. Neural synthesis of binaural speech from mono audio. In *ICLR*, 2020.

[160] Thomas D. Rossing and Neville H. Fletcher. *Physics of Musical Instruments 2nd Edition*. Springer New York, New York, NY, 1998.

[161] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[162] Yulia Rubanova, Ricky TQ Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *NeurIPS*, 2019.

[163] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[164] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909, 2016.

[165] Rebecca Salles, Kele Belloze, Fabio Porto, Pedro H Gonzalez, and Eduardo Ogasawara. Nonstationary time series transformation methods: An experimental review. *Knowledge-Based Systems*, 164:274–291, 2019.

[166] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016.

[167] Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.

[168] Siyuan Shan, Vishal Athreya Baskaran, Haidong Yi, Jolene Ranek, Natalie Stanley, and Junier B. Oliva. Transparent single-cell set classification with kernel mean embeddings. In *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '22, New York, NY, USA, 2022. Association for Computing Machinery.

[169] Siyuan Shan, Lamtharn Hantrakul, Jitong Chen, Matt Avent, and David Trevelyan. Differentiable wavetable synthesis. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4598–4602, 2022.

[170] Siyuan Shan, Yang Li, and Junier B Oliva. Meta-neighborhoods. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[171] Siyuan Shan, Yang Li, and Junier B Oliva. Nrtsi: Non-recurrent time series imputation. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[172] Siyuan Shan and Yi Ren. Automatic audio tagging with 1d and 2d convolutional neural networks technical report. *Detection and Classification of Acoustic Scenes and Events 2018*.

[173] Siyuan Shan, Wen Yan, Xiaoqing Guo, Eric I Chang, Yubo Fan, Yan Xu, et al. Unsupervised end-to-end learning for deformable medical image registration. *arXiv preprint arXiv:1711.08608*, 2017.

[174] Yifeng Shi, Junier Oliva, and Marc Niethammer. Deep message passing on sets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5750–5757, 2020.

[175] Satya Narayan Shukla and Benjamin Marlin. Multi-time attention networks for irregularly sampled time series. In *International Conference on Learning Representations*, 2021.

[176] Julius Smith. *Physical Audio Signal Processing: for Virtual Musical Instruments and Digital Audio Effects*. 01 2006.

[177] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[178] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2006.

[179] Pablo Sprechmann, Siddhant Jayakumar, Jack Rae, Alexander Pritzel, Adria Puigdomenech Badia, Benigno Uria, Oriol Vinyals, Demis Hassabis, Razvan Pascanu, and Charles Blundell. Memory-based parameter adaptation. In *International Conference on Learning Representations*, 2018.

[180] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[181] Natalie Stanley, Ina A Stelzer, et al. Vopo leverages cellular heterogeneity for predictive modeling of single-cell data. *Nature communications*, 11(1):1–9, 2020.

[182] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[183] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[184] Danica J Sutherland and Jeff Schneider. On the error of random fourier features. *arXiv preprint arXiv:1506.02785*, 2015.

[185] Dougal Sutherland, Junier Oliva, Barnabás Póczos, and Jeff Schneider. Linear-time learning on distributions with approximate kernel embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[186] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems*, 34, 2021.

[187] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[188] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574, 2009.

[189] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[190] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[191] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[192] Arun Venkatraman, Martial Hebert, and J Bagnell. Improving multi-step prediction of learned time series models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[193] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *ICLR*, 2016.

[194] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[195] Polina Vishnyakova, Andrey Elchaninov, Timur Fatkhudinov, and Gennady Sukhikh. Role of the Monocyte-Macrophage system in normal pregnancy and preeclampsia. *Int. J. Mol. Sci.*, 20(15), jul 2019.

[196] Vladimir Vovk. Kernel ridge regression. In *Empirical inference*, pages 105–116. Springer, 2013.

[197] Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Ingmar Posner, and Michael Osborne. On the limitations of representing functions on sets. *arXiv preprint arXiv:1901.09006*, 2019.

[198] Jing Wang et al. Assessment efficacy of neutrophil-lymphocyte ratio and monocyte-lymphocyte ratio in preeclampsia. *J. Reprod. Immunol.*, 132:29–34, apr 2019.

[199] Jingyuan Wang, Ze Wang, Jianfeng Li, and Junjie Wu. Multilevel wavelet decomposition network for interpretable time series analysis. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2437–2446, 2018.

[200] Jingyuan Wang, Junjie Wu, Ze Wang, Fei Gao, and Zhang Xiong. Understanding urban dynamics via context-aware tensor factorization with neighboring regularization. *IEEE Transactions on Knowledge and Data Engineering*, 32(11):2269–2283, 2019.

[201] Li Wang, Dong Nie, Guannan Li, Élodie Puybareau, Jose Dolz, Qian Zhang, Fan Wang, Jing Xia, Zhengwang Wu, Jia-Wei Chen, et al. Benchmark on automatic six-month-old infant brain segmentation algorithms: the iseg-2017 challenge. *IEEE transactions on medical imaging*, 38(9):2219–2230, 2019.

[202] Shuai Wang, Wenxuan Wang, Jinming Zhao, Shizhe Chen, Qin Jin, Shilei Zhang, and Yong Qin. Emotion recognition with multimodal features and temporal models. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, pages 598–602, 2017.

[203] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[204] Wenxuan Wang and Zhaopeng Tu. Rethinking the value of transformer components. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6019–6029, 2020.

[205] Ze Wang, Xiuyuan Cheng, Guillermo Sapiro, and Qiang Qiu. Stochastic conditional generative networks with basis decomposition. In *International Conference on Learning Representations*, 2020.

[206] Ze Wang, Sihao Ding, Ying Li, Minming Zhao, Sohini Roychowdhury, Andreas Wallin, Guillermo Sapiro, and Qiang Qiu. Range adaptation for 3d object detection in lidar. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[207] Ze Wang, Zichen Miao, Jun Hu, and Qiang Qiu. Adaptive convolutions with per-pixel dynamic filter atom. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12302–12311, 2021.

[208] Ze Wang, Zichen Miao, Xiantong Zhen, and Qiang Qiu. Learning to learn dense gaussian processes for few-shot learning. *Advances in Neural Information Processing Systems*, 34:13230–13241, 2021.

[209] Ze Wang, Jiang Wang, Zicheng Liu, and Qiang Qiu. Energy-inspired self-supervised pretraining for vision models. In *The Eleventh International Conference on Learning Representations*, 2023.

[210] Ze Wang, Zehao Xiao, Kai Xie, Qiang Qiu, Xiantong Zhen, and Xianbin Cao. In defense of single-column networks for crowd counting. In *British Machine Vision Conference 2018, BMVC 2018, Newcastle, UK, September 3-6, 2018*, page 78. BMVA Press, 2018.

[211] Zifeng Wang, Zheng Zhan, Yifan Gong, Geng Yuan, Wei Niu, Tong Jian, Bin Ren, Stratis Ioannidis, Yanzhi Wang, and Jennifer Dy. Sparcl: Sparse continual learning on the edge. In *Advances in Neural Information Processing Systems*.

[212] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVI*, pages 631–648. Springer, 2022.

[213] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 139–149, 2022.

[214] Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1121–1128, 2009.

[215] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[216] Haoran Wu, Wenxuan Wang, Yuxuan Wan, Wenxiang Jiao, and Michael R Lyu. Chatgpt or grammarly? evaluating chatgpt on grammatical error correction benchmark.

[217] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Self-attention with functional time representation learning. In *Advances in Neural Information Processing Systems*, pages 15915–15925, 2019.

[218] Yan Xu, Siyuan Shan, Ziming Qiu, Zhipeng Jia, Zhengyang Shen, Yipei Wang, Mengfei Shi, and Eric I-Chao Chang. End-to-end subtitle detection and recognition for videos in east asian languages via cnn ensemble. *Signal Processing: Image Communication*, 60:131–143, 2018.

[219] Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 34:17084–17097, 2021.

[220] Hong-Ming Yang, Xu-Yao Zhang, Fei Yin, and Cheng-Lin Liu. Robust classification with convolutional prototype learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3474–3482, 2018.

[221] Haidong Yi and Natalie Stanley. Cytoset: Predicting clinical outcomes via set-modeling of cytometry data. In *Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB '21. Association for Computing Machinery, 2021.

[222] Jinsung Yoon, James Jordon, and Mihaela Schaar. Gain: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning*, pages 5689–5698, 2018.

[223] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in Neural Information Processing Systems*, 30, 2017.

[224] Le Zhang, Yubo Luo, and Shahriar Nirjon. Demo abstract: Capuchin: A neural network model generator for 16-bit microcontrollers. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 497–498. IEEE, 2022.

[225] Qian Zhang, Li Wang, Xiaopeng Zong, Weili Lin, Gang Li, and Dinggang Shen. Frnet: flattened residual network for infant mri skull stripping. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 999–1002. IEEE, 2019.

[226] Shuyi Zhang, Bin Guo, Anlan Dong, Jing He, Ziping Xu, and Song Xi Chen. Cautionary tales on air-quality improvement in beijing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2205):20170457, 2017.

[227] Qilin Zheng, Zongwei Wang, Zishun Feng, Bonan Yan, Yimao Cai, Ru Huang, Yiran Chen, Chia-Lin Yang, and Hai Helen Li. Lattice: An adc/dac-less reram-based processing-in-memory architecture for accelerating deep convolution neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[228] Xiang Zhou, Shiyue Zhang, and Mohit Bansal. Masked part-of-speech model: Does modeling long context help unsupervised pos-tagging? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1099–1114, 2022.