# Formatting Instructions For NeurIPS 2023

**Anonymous Author(s)**
Affiliation
Address
`email`

## Abstract

The abstract paragraph should be indented ½ inch (3 picas) on both the left- and right-hand margins. Use 10 point type, with a vertical spacing (leading) of 11 points. The word **Abstract** must be centered, bold, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

## 1 Submission of papers to NeurIPS 2023

## 2 Introduction

## 3 Diffusion Models

In this section we quckly review the basics of diffusion models. We focus on the stochastic differential equation formulation first presented by **(author?)** [5].

Let $p(\boldsymbol{y})_{\text{data}}$ denote the data distribution. The goal of a diffusion model is to learn a mapping from a simple distribution $p(\boldsymbol{z})$ to the data distribution $p(\boldsymbol{y})_{\text{data}}$.

This is achived by reversing a diffusion process. In particular, we construct a stochastic differential equation $\boldsymbol{y}(t)$ from $t \in [0, T]$ such that $\boldsymbol{y}(0) \sim p(\boldsymbol{y})_{\text{data}}$ and $\boldsymbol{y}(1) \sim p(\boldsymbol{y}(T))$ is a simple distribution we can sample from and whose evolution is given by

$$d\boldsymbol{y}(t) = \boldsymbol{f}(\boldsymbol{y}(t), t)dt + \boldsymbol{g}(t)d\boldsymbol{w}(t), \tag{1}$$

where $\boldsymbol{w}(t)$ is a standard Brownian motion and $\boldsymbol{f} : \mathbb{R}^d \times [0, T] \to \mathbb{R}^d$ and $\boldsymbol{g} : [0, T] \to \mathbb{R}$ are called the drift coefficient and the diffusion coefficient, respectively.

It is possible to reverse this SDE and sample from $p(\boldsymbol{y})_{\text{data}}$ by first sampling $\boldsymbol{y}(1) \sim p(\boldsymbol{y}(T))$ and then evolving the system backwards in time. This is done by solving the reverse SDE (Cite anderson 1982)

$$d\boldsymbol{y}(t) = [f(\boldsymbol{y}(t), t) - g(t)^2 \nabla_{\boldsymbol{y}(t)} \log p(\boldsymbol{y}(t))]dt + g(t)d\bar{\boldsymbol{w}}(t). \tag{2}$$

where $\bar{\boldsymbol{w}}(t)$ is a standard Brownian with reversed time. Thus because $\boldsymbol{f}$ and $g$ are known, and we construct the SDE so that $p(\boldsymbol{y}(T))$ is simple, as long as we know the score $\nabla_{\boldsymbol{y}(t)} \log p(\boldsymbol{y}(t))$ we can sample from $p(\boldsymbol{y})_{\text{data}}$.

#### Estimating the Score

An important result by **(author?)** [6] is that it is possible to estimate the score $\nabla_{\boldsymbol{y}(t)} \log p(\boldsymbol{y}(t))$ by computing

$$\boldsymbol{s}^* = \text{argmin}_{\boldsymbol{s} \in \mathcal{S}} \mathbb{E}_t \mathbb{E}_{p(\boldsymbol{y}(0))_{\text{data}}} \mathbb{E}_{p(\boldsymbol{y}(t)|\boldsymbol{y}(0))} \left[ \left\| \nabla_{\boldsymbol{y}(t)} \log p(\boldsymbol{y}(t)|\boldsymbol{y}(0)) - \boldsymbol{s}(\boldsymbol{y}(t), t) \right\|^2 \right]. \tag{3}$$

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

where $\mathcal{S} = \{\boldsymbol{s} : \mathbb{R}^d \times [0, T] \to \mathbb{R}^d\}$ is the set of all possible score functions indexed by time $t$, and $\mathbb{E}_t$ denotes the expectaion over uniformly sampled $t \in [0, T]$.

**Conditional Diffusion Models**

Although it is most common to train diffusion models unconditionally as explained above, one can also train diffusion models conditionally on some input $\boldsymbol{x}$.

To do so we make the following modifications to the above formulation.

1. We construct one separate SDE per value of $\boldsymbol{x}$. Each SDE shares the same drift and diffusion coefficients but the initial distribution $p_{\boldsymbol{x}}(\boldsymbol{y}(0))$ is given by $p(\boldsymbol{y}|\boldsymbol{x})_{\text{data}}$.
2. The reverse SDE is now given by
$$d\boldsymbol{y}(t) = [f(\boldsymbol{y}(t), t) - g(t)^2 \nabla_{\boldsymbol{y}(t)} \log p_{\boldsymbol{x}}(\boldsymbol{y}(t))]dt + g(t)d\bar{\boldsymbol{w}}(t). \tag{4}$$
where $\nabla_{\boldsymbol{y}(t)} \log p_{\boldsymbol{x}}(\boldsymbol{y}(t))$ is the score of the conditional distribution $p_{\boldsymbol{x}}(\boldsymbol{y}(t))$. Importantly, because we choose the diffusion and drift coefficients so that at $t = T$ the distribution is the same for all values of $\boldsymbol{x}$, we can still sample from the data distribution in the same way as before.
3. The final change is that the score function is now estimated by
$$\boldsymbol{s}^* = \text{argmin}_{\boldsymbol{s} \in \mathcal{S}} \mathbb{E}_t \mathbb{E}_{p(\boldsymbol{y}(0), \boldsymbol{x})_{\text{data}}} \mathbb{E}_{p(\boldsymbol{y}(t)|\boldsymbol{y}(0))} \left[ \left\| \nabla_{\boldsymbol{y}(t)} \log p(\boldsymbol{y}(t)|\boldsymbol{y}(0)) - \boldsymbol{s}(\boldsymbol{y}(t), \boldsymbol{x}, t) \right\|^2 \right].$$
with the changes being that now $\mathcal{S} = \{\boldsymbol{s} : \mathbb{R}^d \times \mathbb{R}^m \times [0, T] \to \mathbb{R}^d\}$ is the set of all possible score functions but now allowing for the score to depend on the input $\boldsymbol{x}$, and the expectation is taken over the joint distribution $p(\boldsymbol{y}(0), \boldsymbol{x})_{\text{data}}$. We emphasize that the score of the conditional distribution $p(\boldsymbol{y}(t)|\boldsymbol{y}(0))$ is still the same because once we condition on $\boldsymbol{y}(0)$ the distribution is the same for all values of $\boldsymbol{x}$.

This formulation of conditional diffusion models is different than controllable generation as presented in [5]. There, a conditional diffusion model is constructed by noting that
$$\nabla_{\boldsymbol{y}(t)} p(\boldsymbol{y}(t)|\boldsymbol{x}) = \nabla_{\boldsymbol{y}(t)} p(\boldsymbol{y}(t)) + \nabla_{\boldsymbol{y}(t)} p(\boldsymbol{x}|\boldsymbol{y}(t))$$
and hence if we obtain the first term from an unconditional diffusion model, and the second term by differentiating through another trained model $p(\boldsymbol{x}|\boldsymbol{y}(t))$, we can obtain the score of the conditional distribution. In our case this is not feasible because in general the dimension of $\boldsymbol{y}$ will be much smaller than the dimension of $\boldsymbol{x}$.

# 4 Gradient Boosted Trees

Gradient Boosted Trees (GBT) [2] are a popular non-parametric machine learning model for function approximation. The objective is to find a function $F : \mathbb{R}^d \to \mathbb{R}$ that minimizes
$$L(F) = \mathbb{E}_{\boldsymbol{x}, y} \left[ l(y, F(\boldsymbol{x})) \right], \tag{5}$$
where $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function, and the expectation is taken over the joint distribution of the input $\boldsymbol{x}$ and the target $y$. It does this by imposing the requirement that $F$ is as a scaled sum of $M$ decision trees $f_m : \mathbb{R}^d \to \mathbb{R}$, i.e.
$$F(\boldsymbol{x}) = \sum_{m=1}^{M} \epsilon f_m(\boldsymbol{x}), \quad \epsilon \in (0, 1). \tag{6}$$
where $\epsilon$ is a learning rate or shrinkage parameter. In the most basic form of the algorithm each tree is constructed to approximate gradient descent on the loss function $L(F)$. In particular, if we let $F_i = \sum_{m=1}^{i} f_m$ denote the function after $i$ iterations and then the $i$-th tree is constructed to approximately minimize the squared error
$$f_i = \text{argmin}_f \mathbb{E}_{\boldsymbol{x}, y} \left( f(\boldsymbol{x}) - \left. \frac{\partial l(y, \hat{y})}{\partial \hat{y}} \right|_{\hat{y} = F_i(\boldsymbol{x})} \right)^2. \tag{7}$$

using empirical risk minimization and a greedy algorithm to construct the tree.

Various modifications to the basic algorithm have been proposed and implemented such as regularization, special ways of optimizing the tree, support for categorical functions, higher order optimization[1, 3, 4]. In this paper we focus on the implementation of GBTs in the LightGBM library [3] with the understanding that the same principles apply to any other GBT implementation.

# 5 Treeffuser/Treeffusion Models

For $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{y} \in \mathbb{R}^m$ the objective of probabilistic predictions is to produce an estimate of the full conditional distribution $\mathbb{P}[\boldsymbol{y}|\boldsymbol{x}]$. This objective is different than in standard regression where the goal is usually to predict $\mathbb{E}[\boldsymbol{y}|\boldsymbol{x}]$.

The most common approach to solve this problem is via parametric models. This procedure assumes that the distribution $\mathbb{P}[\boldsymbol{y}|\boldsymbol{x}]$ can be well approximated by a parametric family of distributions

$$\mathbb{P}[\boldsymbol{y}|\boldsymbol{x}] = p[\boldsymbol{y}|\theta(\boldsymbol{x})],$$

where $p$ is a well known distribution (e.g Gaussian) and $\theta(\boldsymbol{x})$ is a function that maps $\boldsymbol{x}$ to the parameters of the distribution $p$ (e.g. the mean and covariance of a Gaussian). Optimization is then performed by finding the function $\theta(\boldsymbol{x})$ that minimizes a proper-scoring rule such as the negative log-likelihood.

The advantage of this approach is that, because $p$ is often a simple distribution, it is easy to sample from it, compute the log-likelihood, and evaluate its moments. However, if the assumption that $p$ is a good approximation to $\mathbb{P}[\boldsymbol{y}|\boldsymbol{x}]$ is not met, the model can be poorly calibrated, the predictions can be innacurrate and the uncertainty estimates can be unreliable. Moreover, it is often hard to know when these assumptions are met, and it often takes significant expertise to find an appropiate parametric family of distributions.

Non-parametric models, like diffusions have beecome the state-of-the-art for deep generative models. However, they are not as popular for probabilistic predictions and are often not considered for this task, specially when the probabilistic predictions are over tabular data. Coupled with the fact that diffusion models are often computationally expensive to train, and coding scratch requires significant expertise, it is not surprising that they have not become the standard for probabilistic predictions.

To close this gap we propose a new class of models that we call Treeffuser which adapts diffusion models to the task of probabilistic predictions using gradient boosted trees. The benefit of this approach are:

1. Fast training: Gradient boosted trees are fast to train and can be trained on large datasets. Training on CPUs can be done in a matter of seconds.
2. Easy to use: Because the model is non-parametric, it is not necessary to tune hyperparameters of the distribution $p$. This means that practioners don't need to have a deep understanding of the distribution $p$ to use the model.
3. Large sample convergence guarantees: Due to the non-parametric nature of the model and the score-based formulation of the training, the model is guaranteed to converge to the true distribution as the number of samples goes to infinity.
4. Efficient support for probabilistic predictions over multi-dimensional outputs: The model can be used to predict the full conditional distribution $\mathbb{P}[\boldsymbol{y}|\boldsymbol{x}]$ of multi-dimensional outputs in a manner that scales $O(m)$ with the number of dimensions $m$. This is better than the $O(m^2)$ scaling of parametric models. that require the estimation of the covariance matrix.
5. Support for feautures important for tabular data: Native support for training with categorical variables, missing values, etc.

## 5.1 Treeffuser Model

Treeffuser is an algorithm that combines conditional diffusion with gradient boosted trees. The model is constructed by approximating the score function $\nabla_{\boldsymbol{y}(t)} \log p(\boldsymbol{y}(t)|\boldsymbol{x})$ with $m$ gradient boosted trees $s_i : \mathbb{R}^d \times \mathbb{R}^M \times [0, T] \to \mathbb{R}$ that depend on the input $\boldsymbol{x}, t, \boldsymbol{y}(t)$.

$$\boldsymbol{s}(\boldsymbol{y}(t), \boldsymbol{x}, t) = (s_1(\boldsymbol{y}(t), \boldsymbol{x}, t), \ldots, s_m(\boldsymbol{y}(t), \boldsymbol{x}, t)).$$

**Training:** Each gradient boosted tree $s_i$ is trained to minimize the loss function

$$L_i(s) = \mathbb{E}_t \mathbb{E}_{\boldsymbol{y}(0),\boldsymbol{x}} \mathbb{E}_{\boldsymbol{y}(t)|\boldsymbol{y}(0)} \left[ \left( \frac{\partial \log p(\boldsymbol{y}(t)|\boldsymbol{x})}{\partial y_i(t)} - s(\boldsymbol{y}(t), \boldsymbol{x}, t) \right)^2 \right]. \tag{8}$$

Add the term that we divide by using empirical risk minimization. Due to [6] we know that if the gradient boosted tree adequately minimizes this objective it is guaranteed to converge to the true score function. Algorithm **??** describes the training procedure for the Treeffuser model in more detail.

**Sampling:** Maybe add something about sampling from the model.

**Log-likelihood:** Describe the computation of the log-likelihood.

**Other quantities:** As sampling from the model is very easy it is also easy to compute other quantities via monte carlo integration and in principle it is straightforward to compute any quantity of the form $\mathbb{E}_{\boldsymbol{y} \sim p(\boldsymbol{y}|\boldsymbol{x})} f(\boldsymbol{y})$. This includes the mean, variance, quantiles, but also more complicated quantities where a flexible model is needed. We demonstrate this in the experiments using the estimator for causal effects estimation.

# References

References follow the acknowledgments in the camera-ready paper. Use unnumbered first-level heading for the references. Any choice of citation style is acceptable as long as you are consistent. It is permissible to reduce the font size to `small` (9 point) when listing the references. Note that the Reference section does not count towards the page limit.**(author?)** [5]

# References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, August 2016.

[2] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[3] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[4] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features, 2019.

[5] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

[6] Pascal Vincent. A connection between score matching and denoising autoencoders. Technical Report 1358, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, CP 6128, Succ. Centre-Ville, Montréal (QC) H3C 3J7, Canada, December 2010. THIS IS A PREPRINT VERSION OF A NOTE THAT HAS BEEN ACCEPTED FOR PUBLICATION IN NEURAL COMPUTATION.