Mixup Regularization: A Probabilistic Perspective

Yousef El-Laham[1] **Niccolò Dalmasso**[1] **Svitlana Vyetrenko**[1] **Vamsi K. Potluru**[1] **Manuela Veloso**[1]

[1]J.P. Morgan AI Research, New York, NY, USA

Abstract

In recent years, mixup regularization has gained popularity as an effective way to improve the generalization performance of deep learning models by training on convex combinations of training data. While many mixup variants have been explored, the proper adoption of the technique to conditional density estimation and probabilistic machine learning remains relatively unexplored. This work introduces a novel framework for mixup regularization based on probabilistic fusion that is better suited for conditional density estimation tasks. For data distributed according to a member of the exponential family, we show that likelihood functions can be analytically fused using log-linear pooling. We further propose an extension of probabilistic mixup, which allows for fusion of inputs at an arbitrary intermediate layer of the neural network. We provide a theoretical analysis comparing our approach to standard mixup variants. Empirical results on synthetic and real datasets demonstrate the benefits of our proposed framework compared to existing mixup variants.

1 INTRODUCTION

Mixup regularization has become a notable technique for improving generalization in deep learning for supervised learning tasks [Zhang et al., 2018]. Models trained with mixup backpropagate the loss function on random convex combinations of training sample pairs, using a mixing coefficient sampled from a Beta distribution. Theoretically, mixup has been analyzed as a form of *vicinal risk minimization* (VRM) [Chapelle et al., 2000, Zhang et al., 2021, Carratino et al., 2022], contrasting with *empirical risk minimization* (ERM) [Vapnik, 1999]. Empirical results show mixup effectively improves out-of-sample performance while maintaining competitive performance with state of the art models in data modalities beyond tabular data, such as image [Guo et al., 2019b, Liu et al., 2023, Islam et al., 2024, Wang et al., 2024], natural language processing [Guo et al., 2019a, Sun et al., 2020, Zhang et al., 2020], graph [Verma et al., 2021, Han et al., 2022, Jeong et al., 2023] and speech [Tokozume et al., 2018, Zhang et al., 2022]. Overall, many variations have been introduced to address different aspects of the original strategy, including the choice of mixing coefficient distribution, modifications for different data modalities, and label/feature mixing strategies; we refer the reader to Jin et al. [2024] for a comprehensive survey.

In tandem with these developments and the rise of large generative language models, the idea of combining or "fusing" multiple deep learning models has gained attention as a means to improve model capacity [Cai et al., 2025]. Key frameworks include mixture of experts, which partitions the input space and assigns different models to different regions of the space, and product of experts, which combines model outputs by multiplying their probability distributions [Jacobs et al., 1991, Hinton, 2002] and have been successfully incorporated into transformer architectures to train large scale large language models [Lepikhin et al., 2021, Jiang et al., 2024, Liu et al., 2024]. Related to this are deep ensembling and its variants, such as deep Gaussian mixture ensembling, have been adopted to improve performance and robustness [Lakshminarayanan et al., 2017, El-Laham et al., 2023]. These frameworks can be viewed as probabilistic fusion of multiple predictors; see Koliander et al. [2022] for a review on fusing probability density functions.

Our work is motivated by the success of probabilistic fusion, which has not yet found its footing in the mixup literature. Traditionally, mixup fuses data (or their embeddings) directly rather than operating on statistical manifolds, which are represented by random variables and their corresponding density functions. This limits mixup's application to random variables; adapting mixup to more general settings like conditional density estimation or probabilistic machine learning could leverage probabilistic fusion benefits. In this paper, we

introduce *Prob*abilistic *Mix*up (ProbMix), a general framework for handling uncertainty in mixup regularization by adapting the methodology to a probabilistic setting using the idea of probabilistic fusion. As this is the first approach of its kind, we consider supervised learning tasks using both tabular and time series data, and comparing with the most used variants in such settings: (i) vanilla mixup [Zhang et al., 2018], (ii) manifold mixup [Verma et al., 2019, El-Laham et al., 2024], which constructs mixup augmentations on intermediate layers of the neural network and (iii) local mixup [Guo et al., 2019b] which performs mixup augmentations locally based on the assigned class of the data to address the problem of manifold intrusion.

The contributions of this work are:

1. We present a novel reformulation of mixup from a probabilistic perspective called ProbMix, that regularizes an abitrary model by fusing likelihood functions from different training samples. We show that log-linear fusion of likelihoods is analytically tractable for exponential families members, allowing for easy implementation for both classification and regression settings.
2. We propose an extension of ProbMix called M-ProbMix that allows for probabilistic fusion at any intermediate layer of the conditional density estimator.
3. We provide theoretical results showing that for certain choices of the fusion function, mixup and manifold mixup are special cases of ProbMix and M-ProbMix.
4. We demonstrate the competitive or superior performance of ProbMix and M-ProbMix on classification and regression tasks on several real datasets in terms of uncertainty calibration on out-of-sample data.

2 BACKGROUND

2.1 PROBLEM SETTING

This work studies generalization in supervised learning problems from the perspective of statistical learning theory. In supervised learning, we are interested in learning a function $f : x \in \mathcal{X} \subseteq \mathbb{R}^{d_x} \rightarrow y \in \mathcal{Y} \subseteq \mathbb{R}^{d_y}$, for some $d_x, d_y \in \mathbb{N}^+$. Suppose that the underlying random variables X and Y have a joint probability density function (pdf) $(X, Y) \sim p_{\text{data}}(x, y)$. Our goal is to learn a function $f \in \mathcal{F}$ that minimizes the risk $R[f]$ defined as:

$$R[f] = \mathbb{E}[\ell(f(x), y)] \tag{1}$$

$$= \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(x), y) p_{\text{data}}(x, y) \mathrm{d}x\mathrm{d}y, \tag{2}$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ is a loss function that measures the discrepancy between a prediction $\hat{y} = f(x)$ and the true output y. Typically, one restricts f to belong to a parametric family of functions $f_\theta \in \mathcal{F}_\theta$ defined by parameters $\theta \in \Theta$. In this context, the goal is to learn the parameters $\widehat{\theta}$ such that they solve the following optimization problem:

$$\widehat{\theta} = \arg\min_{\theta\in\Theta} R(\theta), \tag{3}$$

where $R(\theta) = \mathbb{E}[\ell(f_\theta(x), y)]$. Since $p_{\text{data}}(x, y)$ is unknown, it is common to find the optimal $\widehat{\theta}$ by minimizing an approximation of (3). The most common approach is ERM, where parameters are learned by minimizing an approximation of the risk using the empirical distribution of a dataset of i.i.d. observations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$:

$$\widehat{\theta}_{\text{ERM}} = \arg\min_{\theta\in\Theta} \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i). \tag{4}$$

The optimization problem in (4) is a result of considering the following empirical approximation to $p_{\text{data}}(x, y)$:

$$p_{\text{data}}(x, y) \approx \frac{1}{n} \sum_{i=1}^{n} \delta_{X,Y}(x_i, y_i), \tag{5}$$

where $\delta_{X,Y}(x, y)$ is used to denote a Dirac measure centered at (x, y). Importantly, for conditional density estimation tasks, when the loss is defined as $\ell(f_\theta(x), y) = -\log p_\theta(y|x)$, where $-\log p_\theta(y|x)$ denotes the negative log-likelihood (NLL), ERM is equivalent to maximum likelihood estimation of the parameters.

Under the assumption that the training data are independent and identically distributed (i.i.d) and reflect the distribution of out-of-sample data, the ERM principle will lead to a model that generalizes well to out-of-sample data in the limit of infinite training data. This is due to the fact that (4) will produce the same result as the true risk minimization problem in (3) as $n \rightarrow \infty$. Typically, neither of these requirements are satisfied in practice; that is to say that: (a) training data are not infinite and in some cases scarcely available, and (b) out-of-sample data may not conform exactly to the distribution of the observed dataset. To improve performance on out-of-sample data, different approximations of $p_{\text{data}}(x, y)$ can be considered to regularize f_θ to have better generalization properties.

2.2 VICINAL RISK MINIMIZATION AND MIXUP

To combat the issue of limited data availability and out-of-sample distribution mismatch, the VRM principle can be utilized. VRM does not directly use the empirical density of the data to approximate the risk, but rather uses a "perturbed" version of it. Let $\tilde{p}_{\nu,\mathcal{D}}(x, y)$ denote a joint pdf called the vicinal distribution which depends on training examples $(x_i, y_i) \in \mathcal{D}$ and potentially some additional hyperparameters ν. Then, under the joint pdf $\tilde{p}_{\nu,\mathcal{D}}(x, y)$, the risk can be approximated as follows:

$$\tilde{R}_\nu(\theta) = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f_\theta(x), y) \tilde{p}_{\nu,\mathcal{D}}(x, y) \mathrm{d}x\mathrm{d}y \tag{6}$$

Note that $\tilde{p}_{\nu,\mathcal{D}}(x,y)$ extends the computation of the risk from the exact values of the pair (x_i, y_i) – as in $p_{\text{data}}(x,y)$ in equation 5 – to a neighborhood of (x_i, y_i). In general, regularization based on data augmentation, adversarial training, and label smoothing can be viewed as a form of VRM. To that end, it can be shown that classical mixup and its variants can be viewed as a form of VRM.

Vanilla Mixup. Mixup is a VRM technique that constructs augmented samples by taking random convex combinations of existing ones. The corresponding vicinal distribution in vanilla mixup is:

$$\tilde{p}_{\alpha,\mathcal{D}}(x,y) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbb{E}_\lambda\left[\delta_{X,Y}(\tilde{x}_{i,j,\lambda}, \tilde{y}_{i,j,\lambda})\right], \quad (7)$$

where λ is a mixing coefficient, usually assumed to follow a beta distribution $\mathcal{B}(\alpha,\alpha)$, with equal shape and scale $\alpha > 0$. We define $\tilde{x}_{i,j,\lambda}$ and $\tilde{y}_{i,j,\lambda}$ as

$$\tilde{x}_{i,j,\lambda} = \lambda x_i + (1-\lambda)x_j \quad (8)$$

$$\tilde{y}_{i,j,\lambda} = \lambda y_i + (1-\lambda)y_j \quad (9)$$

Based on this vicinal distribution, the overall loss function that is minimized in vanilla mixup is the following:

$$\tilde{R}_\alpha^{\text{mix}}(\theta) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbb{E}_\lambda\left[\ell(f_\theta(\tilde{x}_{i,j,\lambda}), \tilde{y}_{i,j,\lambda})\right] \quad (10)$$

As $\alpha \to \infty$, the random variable λ converges to a degenerate random variable such that $\mathbb{P}(\lambda = \frac{1}{2}) = 1$. In contrast, as $\alpha \to 0$, the random variable λ converges to a Bernoulli random variable such that $\mathbb{P}(\lambda = 0) = \mathbb{P}(\lambda = 1) = \frac{1}{2}$, in which case, mixup reduces to ERM as in (4). We refer the reader to Appendix A for a detailed review of manifold mixup and local mixup.

3 OUR METHODOLOGY

In this section, we introduce a general methodology for extending mixup regularization to statistical manifolds which we call ***Prob**abilistic **Mix**up* (ProbMix). We begin by discussing how to apply mixup to the likelihood functions obtained via a generic conditional density estimator for both regression and classification using linear and log-linear fusion. Following the methodology for fusing likelihood functions, we present an extension of our approach, called ***M**anifold **Prob**abilistic **Mix**up* (M-ProbMix), that allows for probabilistic mixup at any intermediate layer by introducing a conditional density mapping at the desired layer. As long as the density mapping is an exponential family member, we show that applying ProbMix to an intermediate layer of a neural network is analytically tractable. Figure 1 provides a summary of the training forward pass of ProbMix as compared to mixup-based approaches.

3.1 DATA GENERATING PROCESS

The key distinction between ProbMix and other mixup regularization techniques is the assumption that the responses y are assumed to have been independently generated from an interpolated conditional density function

$$\tilde{p}_\theta(y|x_i, x_j, \lambda) = g_\lambda^x(p_\theta(y|x_i), p_\theta(y|x_j)) \quad (11)$$

obtained by some fusion function g_λ^x, rather than the conditional density $p_\theta(y|x_{i,j,\lambda})$, which conditions based on interpolated features. Let $p(\lambda;\alpha)$ denote a fixed and known pdf over λ with tunable hyperparameters α and let $\mathcal{G} = (\mathcal{D}, \mathcal{E}, \mathcal{W})$ denote a weighted graph defined over the observed dataset. We consider the following data generating process in ProbMix:

$$\lambda \sim p(\lambda;\alpha), \quad (i,j) \sim \mathcal{G} \quad (12)$$

$$y \sim \tilde{p}_\theta(y|x_i, x_j, \lambda) \quad (13)$$

where we use the notation $(i,j) \sim \mathcal{G}$ to denote the generation of an edge $(i,j) \in \mathcal{E}$ from the graph $\mathcal{G}$ based on the weights $\mathcal{W}$. Furthermore, for a given edge (i,j), we assume that the observed label $\tilde{y}$ has a vicinal density

$$\tilde{y} \sim p_\beta(\tilde{y}|y_i, y_j, \lambda) = g_\lambda^y(s_\beta(\tilde{y}|y_i), s_\beta(\tilde{y}|y_j)), \quad (14)$$

obtained by some fusion function g_λ^y, where $s_\beta(\tilde{y}|y)$ denotes a kernel centered around y with hyperparameter β. To summarize, the modeling assumption of ProbMix differs from vanilla mixup in the following two ways:

1. **Data generating process**: In ProbMix the responses are generated from a fusion of conditional densities (guided by a fusion function g_λ^x) each of which are conditioned on distinct features x_i and x_j, respectively. In vanilla mixup, the responses are generated from a single density conditioned on a mixture of features.
2. **Observed responses**: In ProbMix, the responses are assumed to be latent, but with known density $g_\lambda^y(s_\beta(\tilde{y}|y_i), s_\beta(\tilde{y}|y_j))$. In mixup, the responses are observed and assumed to be a convex combination of the corresponding edge that is being mixed.

3.2 OPTIMIZING MODEL PARAMETERS

The goal of ProbMix is to maximize the expected log-likelihood of the latent observations $\tilde{y}$, which can be expressed using the law of iterated expectations as:

$$\begin{aligned} l(\theta;\alpha,\mathcal{G}) &= \mathbb{E}_{\lambda,\tilde{y}}\left[\log \tilde{p}_\theta(\tilde{y}|x_i, x_j, \lambda)\right] \\ &= \mathbb{E}_\lambda\left[\mathbb{E}_{\tilde{y}}\left[\sum_{(i,j)\in\mathcal{E}} w_{i,j}\log\tilde{p}_\theta(\tilde{y}|x_i,x_j,\lambda)\middle|\lambda\right]\right] \\ &= \sum_{(i,j)\in\mathcal{E}} w_{i,j}\mathbb{E}_\lambda\left[\mathbb{E}_{\tilde{y}}[\log\tilde{p}_\theta(\tilde{y}|x_i,x_j,\lambda)|\lambda]\right], \end{aligned}$$

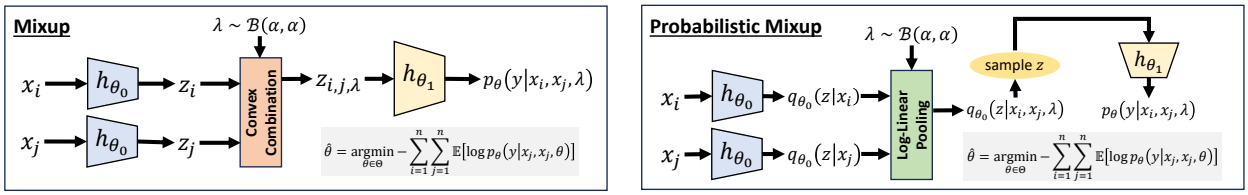


Figure 1: Summary of training forward passes for both mixup vs. ProbMix.

where the inner expectation is taken w.r.t. $p_\beta(\tilde{y}|y_i, y_j, \lambda)$ and the outer expectation is taken w.r.t. $p(\lambda; \alpha)$. Equivalently, we can obtain the corresponding loss function from a VRM perspective by replacing the loss function in (33) with the expected NLL of the fusion function:

$$\tilde{R}^{\mathbb{P}}_{\alpha,\mathcal{G}}(\theta) = -\sum_{(i,j)\in\mathcal{E}} w_{i,j}\mathbb{E}_\lambda[\mathbb{E}_{\tilde{y}}[\log p_\theta(\tilde{y}|x_i, x_j, \lambda)|\lambda]] \quad (15)$$

Depending on the choice of fusion functions g^x_λ and g^y_λ, the overall loss function in (15) will exhibit different regularization effects. We remark that one can also maximize the logarithm of the expected likelihood, which is related to the former by Jensen's inequality. We provide more details about this alternative optimization criterion in Appendix B.

3.2.1 Monte Carlo Approximation

To minimize $\tilde{R}^{\mathbb{P}}_{\alpha,\mathcal{G}}(\theta)$, we can use the Monte Carlo approach to obtain stochastic gradients where instead of taking gradients with respect to $\tilde{R}^{\mathbb{P}}_{\alpha,\mathcal{G}}(\theta)$, we take gradients of an estimator given by:

$$\tilde{R}^{\mathbb{P}}_{\alpha,\mathcal{G}}(\theta) \approx -\sum_{(i,j)\in\mathcal{E}} \frac{w_{i,j}}{K}\sum_{k=1}^{K} \log p_\theta(\tilde{y}^{(k)}|x_i, x_j, \lambda^{(k)}), \quad (16)$$

where $\lambda^{(k)} \sim p(\lambda; \alpha)$ and $\tilde{y}^{(k)} \sim p_\beta(\tilde{y}|y_i, y_j, \lambda^{(k)})$ for $k = 1, \ldots, K$. Typically, a single sample ($K = 1$) is used.

3.2.2 Linear vs. Log-Linear Fusion

The ProbMix framework requires choosing the fusion functions g^x_λ and g^y_λ. Two popular choices often found in the probabilistic fusion literature are linear pooling:

$$g^x_\lambda(p_\theta(y|x_i), p_\theta(y|x_j)) = \lambda p_\theta(y|x_i) + (1-\lambda)p_\theta(y|x_j) \quad (17)$$

and log-linear pooling:

$$g^x_\lambda(p_\theta(y|x_i), p_\theta(y|x_j)) \propto [p_\theta(y|x_i)]^\lambda [p_\theta(y|x_j)]^{1-\lambda} \quad (18)$$

In this work, we utilize the log-linear pooling function, since in the case of exponential family members, the fusion result also belongs to exponential family of probability distributions. Important special cases of this result include the categorical distribution and the Gaussian distribution, which are often the assumed statistical models in classification and regression tasks, respectively. The detailed proof and relevant analytical expressions related to this result can be found in Appendix C.

3.2.3 Why ProbMix over Vanilla Mixup?

Here, we provide an intuitive example demonstrating a scenario where ProbMix is preferred over vanilla mixup. Consider the following data generating process:

$$y = x^3 + (0.5x^2 + 1)\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1), \quad (19)$$

where the ground truth mean and variance functions are $\mu_\theta(x) = x^3$ and $\sigma^2_\theta(x) = (0.5x^2 + 1)^2$, respectively. Suppose two samples $(x_1, y_1) = (5, 130)$ and $(x_2, y_2) = (-5, -120)$ are observed. We would like to understand the regularization effect of both mixup and ProbMix on the ground truth mean and variance functions based on the two observed samples for a mixing coefficient of $\lambda = 0.8$. In the case of vanilla mixup, the predicted conditional is:

$$p_{\text{Mix}}(\tilde{y}|x_1, x_2, \lambda = 0.8) = \mathcal{N}(\tilde{y}|27, 30.25), \quad (20)$$

while for ProbMix it is:

$$p_{\text{ProbMix}}(\tilde{y}|x_1, x_2, \lambda = 0.8) = \mathcal{N}(\tilde{y}|75, 182.25). \quad (21)$$

Assuming $\beta \to 0$ for ProbMix, the interpolated observation $\tilde{y} = \lambda y_1 + (1-\lambda)y_2$ for which the likelihood is evaluated is the same for both approaches and is $\tilde{y} = 80$. Since the interpolated observation $\tilde{y} = 80$ lies in the right tail of p_{Mix}, gradient updates made to the mean and variance functions based on mixup will substantially alter the model, despite the fact that $\mu_\theta(x)$ and $\sigma^2_\theta(x)$ are the ground truth mean and variance functions. In contrast, ProbMix's fused density is much better calibrated to the interpolated target. This demonstrates that in this example, mixup enforces a strong linear bias on the mean and variance functions, which can be inappropriate when the true relationship is nonlinear or when the input features being fused are far apart. By working on the statistical manifold when mixing, ProbMix avoids this strong bias. We provide details of our calculations and conditional density plots related to this illustrative example in Appendix D.

3.3 MANIFOLD PROBABILISTIC MIXUP

We now discuss an extension of ProbMix, called M-ProbMix, that allows for probabilistic fusion in an arbitrary embedding defined by an intermediate layer of the neural network. The idea behind the approach is to consider that our predictor $f_\theta = h_{\theta_1} \circ h_{\theta_0}$ is the composition of the mappings h_{θ_0} and h_{θ_1}. The mapping h_{θ_0} maps the input of the predictor to the parameters of the density of an embedding z (e.g., Gaussian distribution with diagonal covariance matrix), while h_{θ_1} maps from random embedding to the conditional density of the response. Mapping the inputs to a density function in the intermediate layers enables the use of probabilistic fusion to mix samples at the embedding level.

Let $q_{\theta_0}(z|x)$ denote the parametric density of an embedding z given some input feature x. The expected log-likelihood in M-ProbMix can be determined as:

$$l(\theta; \alpha, \mathcal{G}, L_{mix}) = \mathbb{E}[\log \tilde{p}_\theta(y|x_i, x_j, \lambda)] \tag{22}$$

$$= \mathbb{E}_\lambda \left[\mathbb{E}_{\tilde{y}} \left[\sum_{(i,j)\in\mathcal{E}} w_{i,j} \log p_\theta(\tilde{y}|x_i, x_j, \lambda) \middle| \lambda \right] \right], \tag{23}$$

where the density $p_\theta(\tilde{y}|x_i, x_j, \lambda)$ is given by:

$$p_\theta(\tilde{y}|x_i, x_j, \lambda) = \int p_{\theta_1}(\tilde{y}|z) q_{\theta_0}(z|x_i, x_j, \lambda) \mathrm{d}z \tag{24}$$

and the density of z given x_i, x_j and λ is the fusion of the densities of the random embeddings according to some fusion function g^z_λ:

$$q_{\theta_0}(z|x_i, x_j, \lambda) = g^z_\lambda \left(q_{\theta_0}(z|x_i), q_{\theta_0}(z|x_j) \right). \tag{25}$$

We can readily obtain the corresponding risk as:

$$\tilde{R}^{\mathbb{P},\mathcal{M}}_{\alpha,\mathcal{G}}(\theta) = -\sum_{(i,j)\in\mathcal{E}} w_{i,j} \mathbb{E}_\lambda [\mathbb{E}_{\tilde{y}}[\log p_\theta(\tilde{y}|x_i, x_j, \lambda)|\lambda]] \tag{26}$$

Put simply, M-ProbMix can be viewed as a probabilistic extension to manifold mixup, whereby an intermediate layer maps to a random variable rather than a fixed transformation of the input features. Just like ProbMix, the risk of M-ProbMix can be approximated using a Monte Carlo estimate, whereby sampling is additionally done at the embedding level in order to approximate the integral in (24). Importantly, if one chooses $q_{\theta_0}(z|x)$ to belong to a member of the exponential family, then log-linear pooling can be readily applied. A standard and convenient choice is $q_{\theta_0}(z|x)$ is a Gaussian distribution with diagonal covariance matrix, since the reparameterization trick [Kingma, 2013] can be readily applied. Finally, we highlight that the choice of the embedding distribution is agnostic to the learning task, as M-ProbMix focuses on fusing the distributions of the underlying embeddings, rather than the likelihoods themselves.

4 THEORETICAL INSIGHTS

In this section, we provide theoretical insights by comparing the proposed ProbMix and M-ProbMix with mixup and manifold mixup. For simplicity in the presentation, we assume the following:

Assumption 4.1 (Negative Log-Likelihood Loss)**.** The loss function $\ell(f_\theta(x), y) = -\log p_\theta(y|x)$. In the case of classification, $\log p_\theta(y|x)$ corresponds to the log probabilities of each class, while in the case of regression, $\log p_\theta(y|x)$ is assumed to be either a homoscedastic or heteroscedastic Gaussian log-likelihood function.

Assumption 4.2 (Expected Loss over Labels)**.** In the case of classification, label mixing is done by mixing one-hot-encoded vectors, and the risk is taken by taking the expected value over the mixed label probabilities. That is,

$$\mathbb{E}_\lambda[\log p_\theta(y_{i,j,\lambda}|x_{i,j,\lambda})] =$$
$$\mathbb{E}_\lambda[\lambda \log p_\theta(y_i|x_{i,j,\lambda})] + \mathbb{E}_\lambda[(1-\lambda) \log p_\theta(y_j|x_{i,j,\lambda})]$$

In the case of ProbMix, this corresponds to using the following perturbation distribution for the responses:

$$g^y_\lambda(s_\beta(\tilde{y}|y_i), s_\beta(\tilde{y}|y_j)) = \lambda\delta_{y_i} + (1-\lambda)\delta_{y_j}$$

Under Assumption 4.2, the expectation term over the labels for the expected risk of ProbMix and M-ProbMix breaks down into two separate terms due to linearity of expectation. Hence, when comparing mixup and ProbMix across different settings, we can simply focus on comparing log-likelihoods terms, i.e., $\log p_\theta(y|x_{i,j,\lambda})$ for mixup methods and $\log p_\theta(y|x_i, x_j, \lambda)$ for probabilistic mixup methods.

4.1 PROBMIX OPERATES MIXUP ON THE OUTPUTS

Theorems 4.3 and 4.4 show that, under log-linear fusion g^x_λ of the likelihoods, ProbMix can be thought of as operating mixup on the output layers in both classification and regression tasks.

Theorem 4.3 (ProbMix as mixup of Logits)**.** *Under Assumptions 4.2 and 4.1 and log-linear fusion g^x_λ of categorical likelihoods, ProbMix is equivalent to vanilla mixup on the logits.*

Theorem 4.4 (ProbMix as Mixup of Means)**.** *Under Assumptions 4.2 and 4.1 and log-linear fusion g^x_λ of likelihoods, ProbMix is equivalent to vanilla mixup of the output means.*

Proof Sketch.. For both theorems, the proof proceeds constructively showing that the likelihoods of ProbMix and mixup on the output layers are proportional to the same quantities. □

4.2 MIXUP AS A SPECIAL CASE OF PROBMIX

Theorems 4.5 and 4.6 show that in both the classification and regression tasks, ProbMix when using a linear mapping f_θ and log-linear pooling g^x_λ reduces to vanilla mixup. Theorem 4.7 shows that log-linear pooling of homoscedastic Gaussian embeddings makes M-ProbMix reducing to manifold mixup, as long as embedding means are propagated during training. We refer the reader to Appendix E for full proofs.

Theorem 4.5 (Mixup and ProbMix for Multiclass Logistic Regression)**.** *In classification tasks, under Assumptions 4.2 and 4.1, when setting g^x_λ as log-linear pooling of categorical distributions and using a multi-class logistic regression learner f_θ, ProbMix reduces to vanilla Mixup.*

Proof Sketch. Consider $f_\theta = h_1 \circ h_\theta$, where $h_1(z)$ is the softmax function and $h_\theta(x) = Ax + b$ is a linear function with $A = [a_1, \ldots, a_y]^\intercal \in \mathbb{R}^{d_y \times d_x}$ and $b \in \mathbb{R}^{d_y}$. The proof proceeds constructively by showing that the probability of the kth class is identical for both ProbMix in the settings above and vanilla Mixup. □

Theorem 4.6 (Mixup and ProbMix for Linear Regression)**.** *In regression tasks, under Assumptions 4.2 and 4.1, when setting g^x_λ as log-linear pooling of homoscedastic Gaussian distributions and using a linear regression learner $f_\theta(x) = Ax + b$, ProbMix reduces to vanilla mixup.*

Proof Sketch. The proof proceeds constructively by showing that the log-likelihood is proportional to the same quantities for both ProbMix in the settings above and traditional mixup. □

In the case of probabilistic manifold mixup with the choice of the hidden distribution as a homoscedastic Gaussian,

Theorem 4.7 (M-ProbMix under Homoscedastic Gaussian Approximation is Manifold Mixup)**.** *Assume a learner $f_{\theta,\phi} = d_\phi \circ h_\theta$, where h_θ and d_ϕ are encoder and decoder, respectively, and means are propagated directly during both training and inference, i.e., $f_{\theta,\phi} = d_\phi(h_\theta(x))$. Then, under Assumptions 4.2 and 4.1, M-ProbMix using a log-linear fusion a homoscedastic Gaussian embeddings is equivalent to manifold mixup.*

Proof Sketch. As log-linear fusion of Gaussians is Gaussian (see Section 3.2.2 and Appendix C), and the means are propagated during training and inference, one can show the log-likelihoods of M-ProbMix and manifold mixup are equal. □

5 PRACTICAL CONSIDERATIONS

Sampling strategies: When selecting pairs of samples for training, one option is to uniformly sample across all possible pairs in the training data, as in classical mixup. This corresponds to uniformly sampling edges in a fully connected graph where training samples are nodes. However, the edges between a pair of points (x_i, x_j) can be assigned weights $w_{i,j}$ to enforce non-uniform sampling strategies. In this work, we consider two types of sampling graphs: a *fully-connected graph*, such that $w_{i,j} = \frac{1}{n^2}$ for all edges $(i,j) \in \mathcal{E}$; and a *nearest neighbors graph*, where $w_{i,j} \propto 1$ for all $(i,j) \in \mathcal{E}$ such that $x_j \in \mathcal{C}_K(x_i)$, where $\mathcal{C}_K(x_i)$ denotes the set of K nearest neighbors of x_i. We refer to variants of our methods that utilize the fully-connected graph as ProbMix and M-ProbMix and the methods that utilize the nearest neighbors graph as Loc^{K}ProbMix and Loc^{K}M-ProbMix. By default, we utilize $K = 5$ neighbors, unless otherwise specified.

Mixing and perturbation distributions: In this work, we consider $\lambda \sim \mathcal{B}(\alpha, \alpha)$, where $\alpha \in (0, 1)$. With regards to the choice of the response perturbation distribution $s_\beta(\tilde{y}|y_i)$, since we are utilizing log-linear pooling, we need to guarantee that the distribution $s_\beta(\tilde{y}|y_i)$ is positive (i.e., has nonzero probability density over $\mathcal{Y}$).

In regression tasks we use an isotropic Gaussian with variance $\beta > 0$, i.e.,

$$s_\beta(\tilde{y}|y_i) = \mathcal{N}(\tilde{y}|y_i, \beta \mathbf{I}_{d_y}).$$

In classification tasks, we bias the probabilities of each label (treated as a one-hot-encoded vector) by a positive constant $\beta > 0$ such that the resulting distribution satisfies

$$\mathbb{P}(\tilde{y} = k|y_i) \propto \mathbb{P}(y_i = k) + \beta.$$

We note that in classification settings we normalize for the perturbed response distribution to be a valid probability distribution. We conducted ablation studies on both a toy regression and classification dataset to explore different settings of α and β: more details can be found in Appendix F.1.

Mixing layer and embedding distribution: For M-ProbMix, the choice of the embedding and its distributional form are important considerations. In this work, we choose the embedding layer to be the first layer after feature extraction. In the case of simple regression or classification tasks, this could be after the first of second layer of a fully connected network. For more complex architectures, such as LSTMs and transformers, we use the flattened output (potentially transformed to a lower dimensional space) as the embedding layer. In terms of the distributional form, we utilize a diagonal Gaussian parameterized as follows:

$$q_{\theta_0}(z|x) = \mathcal{N}(z|h_{\theta_0}(z), \text{diag}(\sigma^2_{\theta_0})), \tag{27}$$

where $\sigma^2_{\theta_0} : \mathcal{X} \to \mathbb{R}^{d_z}$ is a variance network that maps the input data to the diagonal covariance matrix of the embedding. Importantly, for computational efficiency, $\sigma^2_{\theta_0}$ should share parameters with $h_{\theta_0}(x)$, especially for larger architectures. Finally, we highlight that for this choice of $q_{\theta_0}(z|x)$, the reparameterization trick can be applied to reduce the variance of stochastic gradients during training.

6 EXPERIMENTS

In this section, we provide an empirical evaluation comparing our approaches ProbMix, M-ProbMix, Loc^{K}ProbMix, and Loc^{K}M-ProbMix with ERM and classical variants of mixup regularization: vanilla mixup (Mix), manifold mixup (M-Mix), and local mixup (Loc^{K}Mix). We additionally compare to a combination of manifold mixup and local mixup (Loc^{K}M-Mix). We note that local variants of both ProbMix and Mix assume a nearest neighbors sampling graph with $K = 5$ neighbors. We also split the training dataset into 80% train and 20% validation, unless noted otherwise. Once training is complete, the model parameters that provide the smallest loss on the validation dataset is used for evaluation.

6.1 TOY DATASETS

Toy regression: Consider the following data generating process for the toy regression dataset:

$$y_i = x_i^3 + \epsilon_i, \tag{28}$$

where $\epsilon_i \sim \mathcal{N}(0, 9)$. We generate $n = 100$ training examples such that $x_i \sim \mathcal{U}(-4, 4)$. Test examples are generated by randomly sampling features $x_i \sim \mathcal{U}(4, 6)$, which are considered out-of-distribution with respect to training data. This experimental setup allows us to test the extrapolation capabilities of each method. For each method, we use a two-layer multi-layer perceptron (MLP) with 128 and 64 hidden units per layer and train with full-batch gradient descent for $E = 500$ epochs with a learning rate of $\eta = 0.01$. For manifold-based approaches, we set the mixing layer as the first layer of the MLP (i.e., $d_z = 128$). Figure 2 shows a comparison of the extrapolation performance across the different methods. More detailed box plots showing the performance of each method with respect to both mean-squared error (MSE) and negative log-likelihood (NLL) averaged over 10 runs can be found in Appendix F.1.1.

In this example, while Mix achieves lower MSE than ERM, the NLL is much larger on average, implying worse calibration of the conditional density estimates. The other variants of mixup like M-Mix, Loc^{K}Mix, and Loc^{K}Mix achiever similar MSE and NLL to ERM on average, but with much higher variance. This also implies that these other variants of mixup variation do not add any value to out-of-distribution extrapolation, but rather only introduce variability in training. We see that ProbMix and Loc^{K}ProbMix achieve similar performance to the mixup variants, noting that introduces in both cases of classical mixup regularization and probabilistic mixup, introducing locality in the sampling graph improves performance vastly in terms of NLL. This can be attributed to the fact that without locality, likelihoods for dissimilar (distant) features and their corresponding responses are fused, introducing a large bias. We observe the best performance in terms of bothe MSE and NLL with M-ProbMix and Loc^{K}M-ProbMix methods.

Additionally, we provide results comparing the performance of linear and log-linear pooling function for ProbMix in Appendix F.1.2, as well as a comparison to latent variable modeling without mixup regularization in Appendix F.1.3.

Toy classification: We consider a three class toy classification problem from noisy ring-shaped distribution, where the features $x_i \in \mathbb{R}^2$ of each sample if $y_i = k$ is generated :

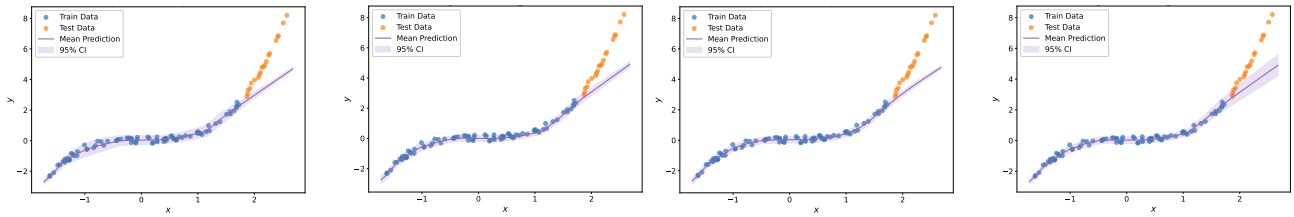
$$x_i = \begin{bmatrix} r_k \cos(\omega_i) \\ r_k \sin(\omega_i) \end{bmatrix} + \epsilon_i, \tag{29}$$

where $\omega_i \sim \mathcal{U}[0, 2\pi]$ and $\epsilon_i \sim \mathcal{N}(0, 0.09\mathbf{I}_2)$. We generate $n = 100$ training examples from this generating process, where approximately an equal number of samples are generated per class with class radii defined by $r_1 = 0.5$, $r_2 = 1.5$, and $r_3 = 2.5$. We generate test samples from the same distribution as training data and utilize the same architecture and training hyperparameters as the toy regression dataset. Detailed results comparing the decision boundaries comparing decision boundaries, test accuracy and NLL, averaged over 10 runs can be found in Appendix F.1.4.

In terms of test accuracy, we observe that all methods perform similarly, with ERM and Loc^{K}M-ProbMix performing best on average. M-Mix and ProbMix (with $\beta = 0$) achieve the worst performance; however, as evidence by the NLL, ProbMix achieves competitive performance in terms of uncertainty calibration. We highlight that M-Mix introduces large bias in the model, as the NLL is about $2\times$ as large as that of ERM. By varying the β parameter, we notice that ProbMix's performance improves in terms of both accuracy and NLL. Our probabilistic mixup framework shows best performance when $\beta = 0.01$, as in this case, all variants of our approach are competitive or outperform baseline approaches.

6.2 UCI REGRESSION

For the regression experiments, we use a two-layer MLP, with 128 and 32 units per layer, respectively. We use a set of regressions datasets from UCI, with a 90%/10% train/test split, and a further 20% of the training data used as validation data to select the best performing model over 1000 epochs. We utilize the 20 train/test splits made available by UCI, as in El-Laham et al. [2023]. The models are trained



(a) Mix. (b) Loc^{K}Mix. (c) ProbMix. (d) Loc^{K}ProbMix.

Figure 2: Visual example of different approaches for toy regression. Mix and Loc^{K}Mix show poor performance on test samples, while Loc^{K}ProbMix, although still mismatching in terms of the mean, outperforms mixup variants in terms of uncertainty calibration on out-of-sample data.

by running the Adam optimizer for $E = 1000$ epochs and a learning rate of $\eta = 0.005$.

Table 1 summarizes the uncertainty quantification results in terms of negative log-likelihood (NLL), while Table 6 in Appendix F provides the prediction performance in terms of root mean-square error (RMSE). Results indicate that ProbMix and its variants improve the uncertainty quantification capabilities of the model in almost all datasets, also providing a better prediction accuracy in terms of RMSE. As we observed the uncertainty quantification generally improves when considering M-ProbMix and various, we have also included a further refinement, which we have indicated as M-ProbMix*. In M-ProbMix*, a separate network is trained to predict the embedding variance, only during inference. For fairness of comparison with other methods, for M-ProbMix* we halved the size of the first embedding, so that the total size of the MLP would be comparable with the other experiments (i.e., the MLP is of size 64 and 32, with an additional variance embedding of size 64). Result show that M-ProbMix* provides competitive results in terms of uncertainty quantification, but predictions performance might degrade in terms of RMSE, likely due to the reduced overall capacity of the prediction model.

6.3 FINANCIAL TIME SERIES FORECASTING

Finally, we demonstrate the performance of probabilistic mixup on a time-series forecasting task. Specifically, we used historical data obtained for the following stocks: Google (GOOG), Gamestop (GME), NVIDIA (NVDA), and Royal Carribean (RCL). [1] . Let $S = [S_1, \dots, S_T] \in \mathbb{R}^{4\times T}$ denote a stock time series, where each S_t corresponds to the open, high, low, and close price for day t. We use the sliding window technique to extract price time-series $x_i = [S_i, \dots, S_{i+W}] \in \mathbb{R}^{4\times W}$ and its corresponding forecast of the close price $y_i = S_{4,i+W+H}$, where $W = 42$ denotes the window size and $H = 21$ denotes the forecast horizon. We utilize data from the period: 01/01/2019 to 01/01/2021 and consider two different time series architectures for this forecasting task: an LSTM architecture with one hidden layer with 64 hidden units; and a transformer architecture with 2 attention heads, 2 encoder/decoder hidden layers with 64 hidden units per layer. For both architectures, the extracted features are flattened and linearly compressed to an embedding of size $d_z = 64$ (which corresponds to the embedding layer). We train the models using the Adam optimizer for $E = 2000$ epochs with a learning rate of $\eta = 0.0005$. Results in terms of NLL can be found for the LSTM and transformer models in Table 2, respectively. Results in terms of test RMSE can be found in Table 7, while the average training NLL and average training RMSE can be found in Tables 8 and 9 in Appendix F.3.

Results show that in 3 out of the 4 stocks, probabilistic mixup variants outperform ERM and standard mixup variants in terms of NLL for both the LSTM and transformer architecture. We can see for the case of GME, both architectures overfit to the training data, reflected in the substantially worse performance. While Mix improves performance as compared to ERM for this dataset, we can see that probabilistic mixup approaches can achieve lower NLL, indicating better calibration of the conditional density estimates on out-of-distribution samples. We attribute this performance gain to the fact that probabilistic mixup approaches provide more conservative predictions of the variance of the conditional density (see sample plots in Appendix F.3).

7 CONCLUSIONS AND FUTURE WORK

In this work, we proposed a novel formulation of mixup regularization tailored for conditional density estimation tasks. Specifically, we introduced a mixup regularization scheme called ProbMix, which operates by probabilistically fusing the conditional densities of different input features within the model. Additionally, we proposed an extension called M-ProbMix, which involves fusing on a statistical manifold defined at an intermediate layer of the network. Our theoretical results demonstrate that many instances of classic mixup regularization can be viewed as specific cases within our proposed framework. Empirical results show that

[1] Data available for download via the yfianace API `https://pypi.org/project/yfinance/`

		Mixup Methods				ProbMixup Methods					
Dataset	ERM	Mix.	Loc^{K}Mix.	M-Mix.	Loc^{K}M-Mix.	ProbMix.	Loc^{K}ProbMix.	M-ProbMix.	Loc^{K}M-ProbMix.	M-ProbMix.*	Loc^{K}M-ProbMix.*
bostonHousing	3.37 ± 0.47	3.32 ± 1.17	3.46 ± 0.76	3.72 ± 2.32	3.31 ± 0.68	3.21 ± 1.07	3.26 ± 0.66	3.12 ± 0.56	3.09 ± 0.70	2.57 ± 0.41	$\mathbf{2.52 \pm 0.21}$
energy	1.24 ± 1.37	1.14 ± 1.00	1.08 ± 1.18	1.25 ± 1.52	1.00 ± 0.88	0.77 ± 0.32	0.82 ± 0.52	$\mathbf{0.74 \pm 0.42}$	0.93 ± 0.65	1.38 ± 0.18	1.13 ± 0.17
wine-quality-red	1.96 ± 2.08	1.84 ± 1.19	1.70 ± 1.43	1.87 ± 2.18	1.99 ± 2.01	1.50 ± 0.57	1.37 ± 0.32	1.26 ± 0.24	1.15 ± 0.18	1.16 ± 0.63	$\mathbf{1.12 \pm 0.22}$
concrete	4.50 ± 2.72	3.71 ± 0.64	3.75 ± 0.77	3.81 ± 0.95	3.69 ± 1.14	3.50 ± 0.58	3.67 ± 0.71	3.35 ± 0.48	3.37 ± 0.47	3.13 ± 0.22	$\mathbf{3.07 \pm 0.16}$
power-plant	2.86 ± 0.18	2.82 ± 0.11	2.84 ± 0.09	2.84 ± 0.12	2.85 ± 0.09	$\mathbf{2.79 \pm 0.06}$	2.84 ± 0.16	2.82 ± 0.06	2.84 ± 0.07	2.85 ± 0.05	2.86 ± 0.05
yacht	0.44 ± 0.94	1.70 ± 0.63	0.78 ± 0.38	0.49 ± 0.66	0.32 ± 0.29	1.39 ± 0.42	1.45 ± 0.88	0.27 ± 0.36	$\mathbf{0.26 \pm 0.25}$	0.92 ± 0.24	1.07 ± 0.28
kin8nm†	-1.14 ± 0.10	-1.01 ± 0.13	-1.14 ± 0.11	-1.16 ± 0.09	-1.07 ± 0.13	-1.12 ± 0.10	-1.15 ± 0.08	$\mathbf{-1.20 \pm 0.07}$	-1.18 ± 0.05	-1.17 ± 0.05	-1.14 ± 0.05
naval-propulsion-plant†	-5.89 ± 0.88	-6.15 ± 0.13	-6.46 ± 0.45	-6.28 ± 0.49	$\mathbf{-6.47 \pm 0.44}$	-6.36 ± 0.63	-6.12 ± 0.89	-6.13 ± 0.30	-5.44 ± 0.63	-5.03 ± 0.04	-4.92 ± 0.06

Table 1: Average NLL for UCI regression datasets. In all datasets (except naval-propulsion-plant), probabilistic mixup variants obtain the best performance as compared to ERM and different mixup variants. († indicates normalization to a single integer digit, while * indicates the use of a separate variance networks, see text for details.)

LSTM		Mixup Methods				ProbMixup Methods			
Dataset	ERM	Mix	Loc^{K}Mix	M-Mix	Loc^{K}M-Mix	ProbMix	Loc^{K}ProbMix	M-ProbMix	Loc^{K}M-ProbMix
GME	158.67 ± 14.80	106.54 ± 5.36	505.33 ± 15.76	170.99 ± 9.25	541.32 ± 50.82	257.74 ± 17.71	411.59 ± 48.22	$\mathbf{67.32 \pm 3.99}$	143.75 ± 8.33
GOOG	36.52 ± 6.26	148.71 ± 21.49	110.14 ± 3.66	102.01 ± 35.98	80.34 ± 15.73	141.25 ± 40.68	52.11 ± 4.27	$\mathbf{5.83 \pm 1.65}$	11.08 ± 2.47
NVDA	3.17 ± 0.73	5.43 ± 1.10	49.79 ± 8.65	9.06 ± 1.93	61.44 ± 15.22	5.19 ± 1.14	61.32 ± 13.26	1.67 ± 0.14	$\mathbf{1.11 \pm 0.08}$
RCL	14.22 ± 0.96	$\mathbf{0.74 \pm 0.08}$	128.31 ± 22.15	61.59 ± 10.45	55.22 ± 6.59	18.32 ± 1.33	56.42 ± 6.33	2.22 ± 0.30	6.22 ± 0.74
Transformer		Mixup Methods				ProbMixup Methods			
Dataset	ERM	Mix	Loc^{K}Mix	M-Mix	Loc^{K}M-Mix	ProbMix	Loc^{K}ProbMix	M-ProbMix	Loc^{K}M-ProbMix
GME	684.79 ± 83.17	507.55 ± 28.04	650.72 ± 59.74	788.60 ± 70.42	551.78 ± 27.83	389.45 ± 17.00	466.82 ± 25.67	$\mathbf{374.15 \pm 41.05}$	398.97 ± 16.66
GOOG	114.32 ± 23.27	147.85 ± 10.57	57.37 ± 16.75	164.20 ± 26.53	64.75 ± 12.21	203.44 ± 31.69	$\mathbf{35.21 \pm 7.98}$	111.34 ± 21.59	86.94 ± 15.56
NVDA	2.29 ± 0.32	1.32 ± 0.16	3.23 ± 0.44	2.38 ± 0.13	1.90 ± 0.11	7.77 ± 0.67	1.78 ± 0.03	$\mathbf{0.91 \pm 0.10}$	1.81 ± 0.08
RCL	10.04 ± 2.17	$\mathbf{-0.05 \pm 0.02}$	10.26 ± 1.17	8.46 ± 1.39	4.85 ± 0.78	7.46 ± 0.55	4.82 ± 0.65	3.46 ± 0.34	4.36 ± 0.85

Table 2: NLL on stock datasets for time series forecasting for LSTM model (above) and transformer model (below). Numerical values have been normalized in each dataset so that the best performing method has one or two integer digits. Probabilistic mixup variants outperforms all other methods except on the the RCL stock, for which Mix performs best.

both ProbMix and M-ProbMix produce conditional density estimates that are significantly better calibrated for out-of-sample data.

While the focus of this work was to devise a framework to extend mixup in the context of conditional density estimations, in future work we plan on extending ProbMix applications to more complex data modalities (such as images, text and speech), as well as including a comparison with existing approaches for quantifying uncertainty in deep learning models, and whether the inclusion of ProbMix could further improve performance (e.g, if every model in the deep ensemble approach in Lakshminarayanan et al. 2017 is trained instead with probabilistic mixup). Potential future research directions include exploring the applicability of our methods to generative frameworks, such as variational autoencoders or denoising diffusion probabilistic models. Finally, we envision a theoretical analysis on the generalization performance of our proposed method could be pursued in a similar fashion as in Yao et al. [2022].

Acknowledgements

The authors would like to thank Nelson Vadori for providing feedback during the research ideation stage.

Disclaimer: This paper was prepared for informational purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co. and its affiliates ("JP Morgan") and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts in large language models. *IEEE Transactions on Knowledge and Data Engineering*, 37(7):3896–3915, 2025. doi: 10.1109/TKDE.2025.3554028.

Luigi Carratino, Moustapha Cissé, Rodolphe Jenatton, and Jean-Philippe Vert. On mixup regularization. *Journal of Machine Learning Research*, 23(325):1–31, 2022.

Olivier Chapelle, Jason Weston, Léon Bottou, and Vladimir Vapnik. Vicinal risk minimization. *Advances in neural information processing systems*, 13, 2000.

Yousef El-Laham, Niccolo Dalmasso, Elizabeth Fons, and

Svitlana Vyetrenko. Deep gaussian mixture ensembles. In *The 39th Conference on Uncertainty in Artificial Intelligence*, 2023. URL https://openreview.net/forum?id=_Qwp7ATIa6P.

Yousef El-Laham, Elizabeth Fons, Dillon Daudert, and Svitlana Vyetrenko. Augment on manifold: Mixup regularization with umap. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7040–7044. IEEE, 2024.

Hongyu Guo, Yongyi Mao, and Richong Zhang. Augmenting data with mixup for sentence classification: An empirical study. arxiv 2019. *arXiv preprint arXiv:1905.08941*, 2019a.

Hongyu Guo, Yongyi Mao, and Richong Zhang. Mixup as locally linear out-of-manifold regularization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3714–3722, 2019b.

Xiaotian Han, Zhimeng Jiang, Ninghao Liu, and Xia Hu. G-mixup: Graph data augmentation for graph classification. In *International Conference on Machine Learning*, pages 8230–8248. PMLR, 2022.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14 (8):1771–1800, 2002.

Khawar Islam, Muhammad Zaigham Zaheer, Arif Mahmood, and Karthik Nandakumar. Diffusemix: Label-preserving data augmentation with diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27621–27630, 2024.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Jongwon Jeong, Hoyeop Lee, Hyui Geon Yoon, Beomyoung Lee, Junhee Heo, Geonsoo Kim, and Kim Jin Seon. igraphmix: Input graph mixup method for node classification. In *The Twelfth International Conference on Learning Representations*, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Xin Jin, Hongyu Zhu, Siyuan Li, Zedong Wang, Zicheng Liu, Chang Yu, Huafeng Qin, and Stan Z Li. A survey on mixup augmentations and beyond. *arXiv preprint arXiv:2409.05202*, 2024.

Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Günther Koliander, Yousef El-Laham, Petar M Djurić, and Franz Hlawatsch. Fusion of probability density functions. *Proceedings of the IEEE*, 110(4):404–453, 2022.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Jihao Liu, Xin Huang, Jinliang Zheng, Yu Liu, and Hongsheng Li. Mixmae: Mixed and masked autoencoder for efficient pretraining of hierarchical vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6252–6261, 2023.

Lichao Sun, Congying Xia, Wenpeng Yin, Tingting Liang, Philip Yu, and Lifang He. Mixup-transformer: Dynamic data augmentation for NLP tasks. In Donia Scott, Nuria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3436–3440, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.305.

Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Between-class learning for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5486–5494, 2018.

Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5): 988–999, 1999.

Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International conference on machine learning*, pages 6438–6447. PMLR, 2019.

Vikas Verma, Meng Qu, Kenji Kawaguchi, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. Graphmix: Improved training of gnns for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10024–10032, 2021.

Zhicai Wang, Longhui Wei, Tan Wang, Heyu Chen, Yanbin Hao, Xiang Wang, Xiangnan He, and Qi Tian. Enhance image classification via inter-class image mixup with diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17223–17233, 2024.

Huaxiu Yao, Yiping Wang, Linjun Zhang, James Y Zou, and Chelsea Finn. C-mixup: Improving generalization in regression. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 3361–3376. Curran Associates, Inc., 2022.

Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL `https://openreview.net/forum?id=r1Ddp1-Rb`.

Linjun Zhang, Zhun Deng, Kenji Kawaguchi, Amirata Ghorbani, and James Zou. How does mixup help with robustness and generalization? In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021. URL `https://openreview.net/forum?id=8yKEo06dKNo`.

Rongzhi Zhang, Yue Yu, and Chao Zhang. SeqMix: Augmenting active sequence labeling via sequence mixup. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8566–8579, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.691. URL `https://aclanthology.org/2020.emnlp-main.691/`.

Xin Zhang, Minho Jin, Roger Cheng, Ruirui Li, Eunjung Han, and Andreas Stolcke. Contrastive-mixup learning for improved speaker verification. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7652–7656. IEEE, 2022.

Mixup Regularization: A Probabilistic Perspective (Appendix)

Yousef El-Laham[1] **Niccolò Dalmasso**[1] **Svitlana Vyetrenko**[1] **Vamsi K. Potluru**[1] **Manuela Veloso**[1]

[1]J.P. Morgan AI Research, New York, NY, USA

A REVIEW OF MIXUP VARIANTS

A.1 MANIFOLD MIXUP.

Manifold mixup is a variant of mixup that instead constructs augmented samples by mixing the features in some hidden layer of the predictor. For discussing manifold mixup, it is helpful to think of the neural network predictor as a composition of two functions $f_\theta = h_{\theta_1} \circ h_{\theta_0}$, where h_{θ_0} is referred to as the *feature extractor* and h_{θ_1} is referred to as the *predictor* and $\theta = \{\theta_0, \theta_1\}$. Mathematically, the loss function for manifold mixup is given by:

$$\tilde{R}^{\mathcal{M}}_{\alpha,L\text{mix}}(\theta) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbb{E}_\lambda\left[\ell(h_{\theta_1}(\tilde{z}_{i,j,\lambda}), \tilde{y}_{i,j,\lambda})\right] \tag{30}$$

where $\tilde{z}_{i,j,\lambda}$ is defined as a mixture of the features in after passing the input features through the feature extractor h_{θ_0}:

$$\tilde{z}_{i,j,\lambda} = \lambda h_{\theta_0}(x_i) + (1-\lambda)h_{\theta_0}(x_j), \tag{31}$$

with the mixing parameter λ following the same setting as in vanilla mixup. We note that manifold mixup introduces an additional hyperparameter, namely in which layer the features are mixed, i.e., how to construct the feature extractor h_{θ_0} and predictor h_{θ_1}. Finally, manifold mixup selects pairs within the manifold by sampling data points uniformly at random, hence inducing a fully connected graph with all edges having the same weights.

A.2 LOCAL MIXUP.

While manifold mixup in essence seems more principled than vanilla mixup, the problem of manifold intrusion can arise in both approaches, as the manifold learned by the feature extractor may not have desirable properties. To combat this, several variants of mixup have been proposed that instead construct mixup augmentations locally based on a weighted graph. Consider a graph $\mathcal{G} = (\mathcal{D}, \mathcal{E}, \mathcal{W})$, where $\mathcal{D}$ are the vertices of the graph, while $\mathcal{E}$ and $\mathcal{W}$ denote edges and weights of the graph, respectively. We use $w_{i,j} \in \mathcal{W}$ to denote the weight of the edge betweeen (x_i, y_i) and (x_j, y_j). For example, local mixup performs vanilla mixup augmentations locally based on the some weighted graph defined over the observed dataset, thereby reducing the risk of mixing samples from different classes that lie on different manifolds. In essence local mixup considers a more general form of the vicinal distribution than vanilla mixup:

$$\tilde{p}_{\alpha,\mathcal{G}}(x,y) = \sum_{(i,j)\in\mathcal{E}} w_{i,j}\mathbb{E}_\lambda\left[\delta_{X,Y}(\tilde{x}_{i,j,\lambda}, \tilde{y}_{i,j,\lambda})\right], \tag{32}$$

which leads to a weighted version of the vanilla mixup loss function:

$$\tilde{R}^{\text{local}}_{\alpha,\mathcal{G}}(\theta) = \sum_{(i,j)\in\mathcal{E}} w_{i,j}\mathbb{E}_\lambda\left[\ell(f_\theta(\tilde{x}_{i,j,\lambda}), \tilde{y}_{i,j,\lambda})\right] \tag{33}$$

This idea can also be trivially extended to manifold mixup.

B LOG-EXPECTED LIKELIHOOD CRITERION

An alternative optimization criterion is to maximize the logarithm of the expected likelihood:

$$l^{\text{up}}(\theta;\alpha,\mathcal{G}) = \log \mathbb{E}\left[\tilde{p}_\theta(\tilde{y}|x_i,x_j,\lambda)\right] \tag{34}$$

$$= \log\left(\sum_{(i,j)\in\mathcal{E}} w_{i,j}\mathbb{E}_\lambda\left[\mathbb{E}_{\tilde{y}}[\tilde{p}_\theta(\tilde{y}|x_i,x_j,\lambda)|\lambda]\right]\right) \tag{35}$$

We note that since the logarithm is outside the expectation operator, this loss function does not have an interpretation from a VRM standpoint. Moreover, it is easy to see that, by Jensen's inequality, $l^{\text{up}}(\theta;\alpha,\mathcal{G})$ upper bounds $l(\theta;\alpha,\mathcal{G})$

$$l^{\text{up}}(\theta;\alpha,\mathcal{G}) = \log \mathbb{E}\left[\tilde{p}_\theta(y|x_i,x_j,\lambda)\right] \tag{36}$$

$$\geq \mathbb{E}\left[\log \tilde{p}_\theta(y|x_i,x_j,\lambda)\right] = l(\theta;\alpha,\mathcal{G}) \tag{37}$$

When $K=1$, we note that Monte Carlo estimates of $l^{\text{up}}(\theta;\alpha,\mathcal{G})$ and $l(\theta;\alpha,\mathcal{G})$ are identical, meaning that from a practical perspective, gradient updates will be the same for both. Importantly, when $K>1$, the expected log-likelihood and logarithm of the expected likelihood will yield different stochastic gradient updates.

C LOG-LINEAR POOLING FOR EXPONENTIAL FAMILY MEMBERS

Theorem C.1 (Log-Linear Pooling of Exponential Families)**.** *Let g^x_λ denote the log-linear pooling function. Suppose that $p_\theta(y|x)$ belongs to the exponential family of probability distributions, that is,*

$$p_\theta(y|x) = h(y)\exp\left(\phi_\theta(x)T(y) - A_\theta(x)\right), \tag{38}$$

Then for two inputs x_i and x_j, the log-linear fusion of $p_\theta(y|x_i)$ and $p_\theta(y|x_j)$ belongs to the same exponential family member. That is,

$$g^x_\lambda(p_\theta(y|x_i),p_\theta(y|x_j)) = h(y)\exp\left(\tilde{\phi}_{\theta,\lambda}(x_i,x_j)T(y) - \tilde{A}_{\theta,\lambda}(x_i,x_j)\right) \tag{39}$$

with

$$\tilde{\phi}_{\theta,\lambda}(x_i,x_j) = \lambda\phi_\theta(x_i) + (1-\lambda)\phi_\theta(x_j) \tag{40}$$

$$\tilde{A}_{\theta,\lambda}(x_i,x_j) = \xi + \lambda A_\theta(x_i) + (1-\lambda)A_\theta(x_j), \tag{41}$$

where ξ denotes a normalization constant.

Proof. In the general case of the likelihood function of an example (x,y) belonging to the exponential family, we have:

$$p_\theta(y|x) = h(y)\exp\left(\phi_\theta(x)T(y) - A(\phi_\theta(x))\right), \tag{42}$$

withe corresponding log-likelihood:

$$\log p_\theta(y|x) = \log h(y) + \phi_\theta(x)T(y) - A(\phi_\theta(x)) \tag{43}$$

A log-linear pooling of the likelihood function of θ for the examples (x_i,y) and (x_j,y) is given by:

$$\log \tilde{p}_\theta(y|x_i,x_j) = \xi(x_i,x_j,\theta) + \lambda\log p_\theta(y|x_i) + (1-\lambda)\log p_\theta(y|x_j) \tag{44}$$

$$= \xi(x_i,x_j,\theta) + \lambda\left(\log h(y) + \phi_\theta(x_i)T(y) - A(\phi_\theta(x_i))\right) + (1-\lambda)\left(\log h(y) + \phi_\theta(x_j)T(y) - A(\phi_\theta(x_j))\right) \tag{45}$$

$$= \xi(x_i,x_j,\theta) + \log h(y) + \left(\lambda\phi_\theta(x_i) + (1-\lambda)\phi_\theta(x_j)\right)T(y) - \left(\lambda A(\phi_\theta(x_i)) + (1-\lambda)A(\phi_\theta(x_j))\right) \tag{46}$$

$$= \log h(y) + \tilde{\phi}_\theta(x_i,x_j,\lambda)T(y) - \tilde{A}(\phi_\theta(x_i),\phi_\theta(x_j),\lambda) \tag{47}$$

where we define:

$$\tilde{\phi}_\theta(x_i,x_j,\lambda) = \lambda\phi_\theta(x_i) + (1-\lambda)\phi_\theta(x_j) \tag{48}$$

$$\tilde{A}(\phi_\theta(x_i),\phi_\theta(x_j),\lambda) = \lambda A(\phi_\theta(x_i)) + (1-\lambda)A(\phi_\theta(x_j)) - \xi(x_i,x_j,\theta) \tag{49}$$

Thus, in general, the log-linear pooling function applied to two exponential family members from the same family results in a exponential family member. Furthermore, since the sufficient statistic $T(y)$ is preserved in the fusion result, the log-linear pooling function will always yield the same exponential family family member. Many important distributions belong to the exponential family of probability distributions. These include normal distributions, beta, gamma, categorical, and Poisson distributions, amongst others. □

C.1 LOG-LINEAR POOLING FOR GAUSSIAN REGRESSION

In the classification setting, we have the following likelihood function for each example (x, y):

$$p_\theta(y|x) = \frac{1}{\sqrt{2\pi\sigma_\theta^2(x)}} \exp\left(-\frac{(y-\mu_\theta(x))^2}{2\sigma_\theta^2(x)}\right) \tag{50}$$

with corresponding log-likelihood function:

$$\log p_\theta(y|x) = -\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\sigma_\theta^2(x) - \frac{(y-\mu_\theta(x))^2}{2\sigma_\theta^2(x)} \tag{51}$$

A log-linear pooling of the likelihood function of θ for the examples (x_i, y) and (x_j, y) is given by:

$$\log\tilde{p}_\theta(y|x_i, x_j) = \xi_0(x_i, x_j, \theta) + \lambda\log p_\theta(y|x_i) + (1-\lambda)\log p_\theta(y|x_j) \tag{52}$$

$$= \xi_1(x_i, x_j, \theta) - \frac{\lambda}{2}\log\sigma_\theta^2(x_i) - \frac{\lambda(y-\mu_\theta(x_i))^2}{2\sigma_\theta^2(x_i)} - \frac{1-\lambda}{2}\log\sigma_\theta^2(x_j) - \frac{(1-\lambda)(y-\mu_\theta(x_i))^2}{2\sigma_\theta^2(x_i)} \tag{53}$$

$$= \xi_1(x_i, x_j, \theta) - \frac{1}{2}\log\left(\left(\sigma_\theta^2(x_i)\right)^\lambda\left(\sigma_\theta^2(x_j)\right)^{1-\lambda}\right) - \frac{\lambda\sigma_\theta^2(x_j)(y-\mu_\theta(x_i))^2 + (1-\lambda)\sigma_\theta^2(x_i)(y-\mu_\theta(x_j))^2}{2\sigma_\theta^2(x_i)\sigma_\theta^2(x_j)} \tag{54}$$

$$\implies \tag{55}$$

$$\tilde{p}_\theta(y|x_i, x_j) = \mathcal{N}\left(y \middle| \left(\frac{\lambda}{\sigma_\theta^2(x_i)} + \frac{1-\lambda}{\sigma_\theta^2(x_j)}\right)^{-1}\left(\lambda\left(\frac{\mu_\theta(x_i)}{\sigma_\theta^2(x_i)}\right) + (1-\lambda)\left(\frac{\mu_\theta(x_j)}{\sigma_\theta^2(x_j)}\right)\right), \left(\frac{\lambda}{\sigma_\theta^2(x_i)} + \frac{1-\lambda}{\sigma_\theta^2(x_j)}\right)^{-1}\right) \tag{56}$$

C.2 LOG-LINEAR POOLING FOR CLASSIFICATION

In the classification setting, we have the following likelihood function for each example (x, y):

$$p_\theta(y|x) = \prod_{k=1}^{K} \pi_{k,\theta}(x)^{\mathbf{1}(y=c_k)} \tag{57}$$

with corresponding log-likelihood:

$$\log p_\theta(y|x) = \sum_{k=1}^{K} \mathbf{1}(y=c_k)\log\pi_{k,\theta}(x) \tag{58}$$

A log-linear pooling of the likelihood function of θ for the examples (x_i, y) and (x_j, y) is given by:

$$\log\tilde{p}_\theta(y|x_i, x_j) = \xi(x_i, x_j, \theta) + \lambda\log p_\theta(y|x_i) + (1-\lambda)\log p_\theta(y|x_j) \tag{59}$$

$$= \xi(x_i, x_j, \theta) + \lambda\left(\sum_{k=1}^{K}\mathbf{1}(y=c_k)\log\pi_{k,\theta}(x_i)\right) + (1-\lambda)\left(\sum_{k=1}^{K}\mathbf{1}(y=c_k)\log\pi_{k,\theta}(x_j)\right) \tag{60}$$

$$= \xi(x_i, x_j, \theta) + \left(\sum_{k=1}^{K}\mathbf{1}(y=c_k)\log\pi_{k,\theta}^{\lambda}(x_i)\right) + \left(\sum_{k=1}^{K}\mathbf{1}(y=c_k)\log\pi_{k,\theta}^{1-\lambda}(x_j)\right) \tag{61}$$

$$= \xi(x_i, x_j, \theta) + \sum_{k=1}^{K}\mathbf{1}(y=c_k)\left(\log\pi_{k,\theta}^{\lambda}(x_i) + \log\pi_{k,\theta}^{1-\lambda}(x_j)\right), \tag{62}$$

where $\xi(x_i, x_j, \theta)$ is a normalizing constant that depends on x_i, x_j, and θ.This result implies that log-linear pooling of two categorical distributions is itself a categorical distribution. In practice, one would just need to combine the logits using a weighted arithmetic average to obtain the fused likelihood function.

D ILLUSTRATIVE EXAMPLE: HETEROSCEDASTIC GAUSSIAN REGRESSION

Here, we elaborate on the illustrative example from Section 3.2.3. In this illustrative example, we consider heteroscedastic Gaussian predictors of the form:

$$y = \mu_\theta(x) + \sqrt{\sigma^2_\theta(x)}\epsilon, \quad \epsilon \sim \mathcal{N}(0,1), \tag{63}$$

where $\mu_\theta(x)$ denotes the *mean function* and $\sigma^2_\theta(x)$ denotes the *variance function*. For two inputs (x_i, y_i) and (x_j, y_j), vanilla mixup regularizes the model by learning to calibrate the random interpolations $\tilde{y} = \lambda y_i + (1-\lambda)y_j$ to the following:

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j \tag{64}$$

$$p_\theta(\tilde{y}|\tilde{x}) = \mathcal{N}(\tilde{y}|\mu_\theta(\tilde{x}), \sigma^2_\theta(\tilde{x})) \tag{65}$$

Note that in Mixup, the input to both μ_θ (mean function) and σ^2_θ (variance function) is the interpolated feature $\tilde{x} = \lambda x_i + (1-\lambda)x_j$. Essentially, vanilla mixup biases both the mean function $\mu_\theta(x)$ and variance function $\sigma^2_\theta(x)$ to behave linearly between observed samples. In contrast, ProbMix learns to calibrate the random interpolations $\tilde{y}$ to a (log-linear) fusion of the predicted conditional densities directly:

$$p_\theta(\tilde{y}|x_i, x_j, \lambda) = \mathcal{N}(\tilde{y}|\mu_\star, \sigma^2_\star) \tag{66}$$

where

$$\sigma^2_\star = \left(\frac{\lambda}{\sigma^2_\theta(x_i)} + \frac{1-\lambda}{\sigma^2_\theta(x_j)}\right)^{-1} \tag{67}$$

$$\mu_\star = \sigma^2_\star\left(\lambda\left(\frac{\mu_\theta(x_i)}{\sigma^2_\theta(x_i)}\right) + (1-\lambda)\left(\frac{\mu_\theta(x_j)}{\sigma^2_\theta(x_j)}\right)\right) \tag{68}$$

ProbMix combines statistical information based on two sources of information (x_i and x_j) and depending on the degree of confidence placed in each source (determined by the value of λ that is sampled), we obtain a different aggregated prediction (fused conditional density). While vanilla mixup biases the mean and variance functions to behave linearly between samples, ProbMix instead biases the model so that predicted conditional densities themselves behave log-linearly between samples.

In the context of the numerical example provided in Section 3.2.3, we consider the following data generating process:

$$y = x^3 + (0.5x^2 + 1)\epsilon, \quad \epsilon \sim \mathcal{N}(0,1). \tag{69}$$

A perfectly fit model would learn mean and variance functions such that $\mu_\theta(x) = x^3$ and $\sigma^2_\theta(x) = (0.5x^2+1)^2$. Supposing that we observe two samples $(x_1, y_1) = (5, 130)$ and $(x_2, y_2) = (-5, -120)$, we would like to understand the regularization effect of both mixup and ProbMix. Consider a sample of the mixing coefficient $\lambda = 0.8$. For this value of λ, vanilla mixup's mean and variance functions are determined by the interpolated feature $\tilde{x} = 0.8 \times 5 + 0.2 \times -5 = 3$, and are given by

$$\mu_\theta(\tilde{x}) = \mu_\theta(3) = 3^3 = 27 \tag{70}$$

$$\sigma^2_\theta(\tilde{x}) = \sigma^2_\theta(3) = (0.5 \times 3^2 + 1)^2 = 5.5^2 = 30.25 \tag{71}$$

and thus the predicted conditional density given by:

$$p_{\mathsf{Mix}}(\tilde{y}|x_1, x_2, \lambda = 0.8) = \mathcal{N}(\tilde{y}|27, 30.25). \tag{72}$$

In ProbMix, the conditional densities are fused and result in another Gaussian conditional density, with variance and mean given by:

$$\sigma^2_\star = \left(\frac{0.8}{\sigma^2_\theta(5)} + \frac{0.2}{\sigma^2_\theta(-5)}\right)^{-1} = \left(\frac{0.8}{182.25} + \frac{0.2}{182.25}\right)^{-1} = 182.25 \tag{73}$$

$$\mu_\star = \sigma^2_\star\left(0.8\left(\frac{\mu_\theta(5)}{\sigma^2_\theta(5)}\right) + 0.2\left(\frac{\mu_\theta(-5)}{\sigma^2_\theta(-5)}\right)\right) = 182.25\left(0.8\left(\frac{125}{182.25}\right) + 0.2\left(\frac{-125}{182.25}\right)\right) = 75 \tag{74}$$

and thus,

$$p_{\mathsf{ProbMix}}(\tilde{y}|x_1, x_2, \lambda = 0.8) = \mathcal{N}(\tilde{y}|75, 182.25), \tag{75}$$

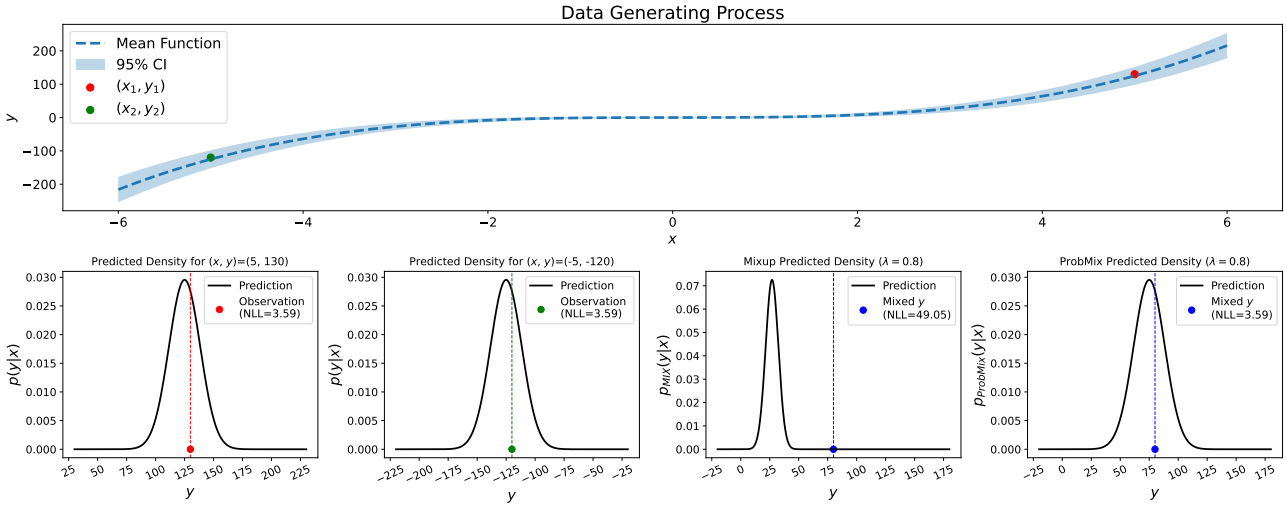


Figure 3: **Top panel**: Data generating process considered in the illustrative example with mean function (blue dashed) and 95% uncertainty band (blue shaded between 2.5% and 97.5% percentiles). **Bottom panel far left**: Conditional density for the input $x = 5$ (under $y = 130$, NLL≈ 3.59). **Bottom panel center left**: Conditional density for the input $x = -5$ (under $y = 120$, NLL≈ 3.59). **Bottom panel center right**: Conditional density for vanilla mixup (under $\tilde{y} = 80$, NLL≈ 49.05).**Bottom panel far right**: Conditional density for ProbMix (under $\tilde{y} = 80$, NLL≈ 3.59).

Taking a perturbation of $\beta \to 0$ for the latent observations, we can consider that both mixup and ProbMix evaluate the likelihood function of the model parameters for an observation of $\tilde{y} = \lambda y_1 + (1-\lambda)y_2 = 80$. The negative log-likelihood of the model parameters in this setting is given by:

$$-\log p_{\text{Mix}}(\tilde{y} = 80 | x_1, x_2, \lambda = 0.8) \approx 49.05 \tag{76}$$

$$-\log p_{\text{ProbMix}}(\tilde{y} = 80 | x_1, x_2, \lambda = 0.8) \approx 3.59 \tag{77}$$

Since the value of the interpolated observation lies in the right tail of p_{Mix}, gradient updates based on mixup will substantially alter the model parameters, despite the fact that $\mu_\theta(x)$ and $\sigma^2_\theta(x)$ are already the ground truth functions. In contrast, ProbMix's fused density is better calibrated to the interpolated observation $\tilde{y} = 80$, leading to more stable and appropriate gradient updates. This demonstrates that mixup enforces a strong linear bias on the mean and variance functions, which becomes problematic when the true relationship is nonlinear or when the input features being mixed are relatively far apart. By mixing on a statistical manifold rather than the input space of the features, ProbMix avoids over-biasing the model. We provide plots of the data generating process, along with the conditional densities of both mixup and ProbMix in Figure 3.

E THEORETICAL INSIGHTS

This section lists the proofs for the theorems in Section 4. We first include the form of the expected risk for mixup, manifold mixup, ProbMix and M-ProbMix for completeness.

Mixup:

$$\tilde{R}^{\text{mix}}_{\alpha,\mathcal{G}}(\theta) = -\sum_{(i,j)\in\mathcal{E}} w_{i,j}\mathbb{E}_\lambda\left[\log p_\theta(y_{i,j,\lambda}|x_{i,j,\lambda})\right],$$

where $y_{i,j,\lambda} = \lambda y_i + (1-\lambda)y_j$ and $x_{i,j,\lambda} = \lambda x_i + (1-\lambda)x_j$.

Manifold Mixup:

$$\tilde{R}^{\mathcal{M}}_{\alpha,L_{\text{mix}}}(\theta) = -\sum_{(i,j)\in\mathcal{E}} w_{i,j}\mathbb{E}_\lambda\left[\log p_{\theta_1}(y_{i,j,\lambda}|z_{i,j,\lambda})\right],$$

where $y_{i,j,\lambda} = \lambda y_i + (1-\lambda)y_j$ and $z_{i,j,\lambda} = \lambda h_{\theta_0}(x_i) + (1-\lambda)h_{\theta_0}(x_j)$.

ProbMix:

$$\tilde{R}^{\mathbb{P}}_{\alpha,\mathcal{G}}(\theta) = -\sum_{(i,j)\in\mathcal{E}} w_{i,j}\mathbb{E}_{\lambda}[\mathbb{E}_{y_{i,j}}[\log p_\theta(y_{i,j}|x_i,x_j,\lambda)]]$$

where $p_\theta(y_{i,j}|x_i,x_j,\lambda) = g^x_\lambda(p_\theta(y_{i,j}|x_i), p_\theta(y_{i,j}|x_j))$

M-ProbMix

$$\tilde{R}^{\mathbb{P},\mathcal{M}}_{\alpha,\mathcal{G}}(\theta) = -\sum_{(i,j)\in\mathcal{E}} w_{i,j}\mathbb{E}_{\lambda}[\mathbb{E}_{y_{i,j}}[\log p_\theta(y_{i,j}|x_i,x_j,\lambda)]]$$

where $p_\theta(y_{i,j}|x_i,x_j,\lambda)$ is given by:

$$p_\theta(y_{i,j}|x_i,x_j,\lambda) = \int p_{\theta_1}(y_{i,j}|z_{i,j})q_{\theta_0}(z_{i,j}|x_i,x_j,\lambda)dz_{i,j}$$

and $q_{\theta_0}(z_{i,j}|x_i,x_j,\lambda) = g^z_\lambda(q_{\theta_0}(z_i|x_i), q_{\theta_0}(z_j|x_j))$ for some fusion function g^z_λ.

Proof of Theorem 4.3. We show the likelihoods of the two approaches are proportional to the same quantity, hence equivalent. Without loss of generality, consider the likelihood for the kth class. For ProbMix, the likelihood is proportional to:

$$\begin{aligned}
\log p^k_\theta(y|x_{i,j,\lambda}) &= \log g^x_\lambda(p^k_\theta(y|x_i), p^k_\theta(y|x_j)) \\
&= \lambda\log(p^k_\theta(y|x_i)) + (1-\lambda)\log(p^k_\theta(y|x_j) \\
&= \lambda\log\left[\frac{e^{-f^k_\theta(x_i)}}{\sum_l e^{-f^l_\theta(x_i)}}\right] + (1-\lambda)\log\left[\frac{e^{-f^k_\theta(x_j)}}{\sum_l e^{-f^l_\theta(x_j)}}\right] \\
&\propto \lambda\log\left(e^{-f^k_\theta(x_i)}\right) + (1-\lambda)\log\left(e^{-f^k_\theta(x_j)}\right)
\end{aligned}$$

Now, considering mixup on the logits, we obtain:

$$\begin{aligned}
\log p_\theta(y|x_{i,j,\lambda}) &= \log p_\theta(y|\lambda f_\theta(x_i) + (1-\lambda)f_\theta(x_j)) \\
&= \log\left(\frac{e^{-\lambda f^k_\theta(x_i)-(1-\lambda)f^k_\theta(x_j)}}{\sum_l e^{-\lambda f^l_\theta(x_i)-(1-\lambda)f^l_\theta(x_j)}}\right) \\
&\propto \log\left[\left(e^{-f^k_\theta(x_i)}\right)^\lambda\right] + \log\left[\left(e^{-f^k_\theta(x_j)}\right)^{(1-\lambda)}\right] \\
&= \lambda\log\left(e^{-f^k_\theta(x_i)}\right) + (1-\lambda)\log\left(e^{-f^k_\theta(x_j)}\right)
\end{aligned}$$

□

Proof of Theorem 4.4. The key is to show the likelihood of both settings is equivalent. Considering ProbMix under log-linear fusion of homoscedastic Gaussian likelihoods, we have:

$$\begin{aligned}
\log p_\theta(y|x_{i,j,\lambda}) &= \log g^x_\lambda(p_\theta(y|x_i), p_\theta(y|x_j)) \\
&= \log\left(\mathcal{N}(y|f_\theta(x_i),\sigma^2\mathbf{I}_{d_y})^\lambda \cdot \mathcal{N}(y|f_\theta(x_j),\sigma^2\mathbf{I}_{d_y})^{(1-\lambda)}\right) \\
&\propto -\lambda\frac{(y-f_\theta(x_i))^T(y-f_\theta(x_i))}{2\sigma^2} - \quad (1-\lambda)\frac{(y-f_\theta(x_j))^T(y-f_\theta(x_j))}{2\sigma^2} \\
&\propto y^Ty - 2y^T\left(\lambda f_\theta(x_i) + (1-\lambda)f_\theta(x_j)\right)
\end{aligned}$$

Now, considering the likelihood for traditional mixup when interpolating the output layer:

$$\begin{aligned}\log p_\theta(y|x_{i,j,\lambda}) &= \log p_\theta(y|\lambda f_\theta(x_i) + (1-\lambda) f_\theta(x_j)) \\ &= C_\sigma^{d_y} - \Big[\frac{(y - \lambda f_\theta(x_i) - (1-\lambda) f_\theta(x_j))^T \quad (y - \lambda f_\theta(x_i) - (1-\lambda) f_\theta(x_j))}{2\sigma^2}\Big] \\ &\propto y^T y - 2y^T \left(\lambda f_\theta(x_i) + (1-\lambda) f_\theta(x_j)\right)\end{aligned}$$

□

Proof of Theorem 4.5. Consider $f_\theta = h_1 \circ h_\theta$, where $h_1(z) = \left[\frac{e^{-z_1}}{\sum_{j=1}^{d_y} e^{-z_j}}, \ldots, \frac{e^{-z_{d_y}}}{\sum_{j=1}^{d_y} e^{-z_j}}\right]^\intercal$ is the softmax function and $h_\theta(x) = Ax + b$ is a linear function with $A = [a_1, \ldots, a_y]^\intercal \in \mathbb{R}^{d_y \times d_x}$ and $b \in \mathbb{R}^{d_y}$. For two inputs x_i and x_j, the probability of the kth class, denoted by $p_{i,k}$ and $p_{j,k}$, respectively, can readily be determined as:

$$p_{i,k} = \frac{e^{-(a_k^\intercal x_i + b_k)}}{\sum_{k'=1}^{d_y} e^{-(a_{k'}^\intercal x_i + b_{k'})}} \tag{78}$$

$$p_{j,k} = \frac{e^{-(a_k^\intercal x_j + b_k)}}{\sum_{k'=1}^{d_y} e^{-(a_{k'}^\intercal x_j + b_{k'})}} \tag{79}$$

For the mixup approach, consider the mixed input $x_{i,j,\lambda} = \lambda x_i + (1-\lambda) x_j$. The output of f_θ is given by:

$$f_\theta(x_{i,j,\lambda}) = h_1(h_\theta(x_{i,j,\lambda})) \tag{80}$$

$$= h_1\left(Ax_{i,j,\lambda} + b\right) \tag{81}$$

$$= h_1(\underbrace{\lambda A x_i + (1-\lambda) A x_j + b}_{z_{i,j,\lambda}}) \tag{82}$$

For each dimension $k = 1, \ldots, d_y$ of $z_{i,j,\lambda}$, denoted by $z_{i,j,\lambda,k} = \lambda a_k^\intercal x_i + (1-\lambda) a_k^\intercal x_j + b_k$ we have that the output of h_1, which is the probability of class k, is given by:

$$p_k^{\text{mix}} = h_1(z_{i,j,\lambda,k}) \tag{83}$$

$$\propto e^{-z_{i,j,\lambda,k}} \tag{84}$$

$$= e^{-(\lambda a_k^\intercal x_i + (1-\lambda) a_k^\intercal x_j + b_k)} \tag{85}$$

$$= e^{-\lambda a_k^\intercal x_i} e^{-(1-\lambda) a_k^\intercal x_j} e^{-b_k} \tag{86}$$

$$= e^{-\lambda(a_k^\intercal x_i + b_k)} e^{-(1-\lambda)(a_k^\intercal x_j + b_k)} \tag{87}$$

$$= \left[e^{-(a_k^\intercal x_i + b_k)}\right]^\lambda \left[e^{-(a_k^\intercal x_j + b_k)}\right]^{1-\lambda} \tag{88}$$

$$\propto \left[h_1(z_{i,k})\right]^\lambda \left[h_1(z_{j,k})\right]^{1-\lambda} \tag{89}$$

$$= p_{i,k}^\lambda p_{j,k}^{1-\lambda} \tag{90}$$

Clearly $p_k^{\text{mix}} \propto p_{i,k}^\lambda p_{j,k}^{1-\lambda}$. Therefore, in this model setting, mixup and ProbMix (under log-linear pooling) are equivalent. □

Proof of Theorem 4.6. Consider that $f_\theta(x) = Ax + b$, where $A \in \mathbb{R}^{d_y \times d_x}$ and $b \in \mathbb{R}^{d_y}$. For two inputs x_i and x_j, the log-likelihood function of θ based on an observed label y is given by:

$$\log p_\theta(y|x_i) = \log \mathcal{N}(y|f_\theta(x_i), \sigma^2 \mathbf{I}_{d_y}) \tag{91}$$

$$\log p_\theta(y|x_j) = \log \mathcal{N}(y|f_\theta(x_j), \sigma^2 \mathbf{I}_{d_y}) \tag{92}$$

where so we have that:

$$\log p_\theta(y|x) = C_\sigma^{d_y} - \frac{(y - f_\theta(x))^\intercal (y - f_\theta(x))}{2\sigma^2}, \tag{93}$$

where $C_\sigma^{d_y} = -\frac{d_y}{2}\log 2\pi\sigma^2$. For mixup, consider mixed input $x_{i,j,\lambda} = \lambda x_i + (1-\lambda)x_j$. Then, the log-likelihood function of an observation y is given by:

$$\log p_\theta(y|x_{i,j,\lambda}) = \log p_\theta(y|\lambda x_i + (1-\lambda)x_j) \tag{94}$$

$$= C_\sigma^{d_y} - \frac{(y - f_\theta(x_{i,j,\lambda}))^\intercal (y - f_\theta(x_{i,j,\lambda}))}{2\sigma^2} \tag{95}$$

We can expand the quadratic as:

$$\begin{aligned}(y - f_\theta(x_{i,j,\lambda}))^\intercal (y - f_\theta(x_{i,j,\lambda})) =& y^\intercal y - 2\underbrace{y^\intercal f(x_{i,j,\lambda})}_{y^\intercal A x_{i,j,\lambda} + y^\intercal b} + \underbrace{f(x_{i,j,\lambda})^\intercal f(x_{i,j,\lambda})}_{x_{i,j,\lambda}^\intercal A^\intercal A x_{i,j,\lambda} + 2b^\intercal A x_{i,j,\lambda} + b^\intercal b} \\ &= y^\intercal y - 2y^\intercal A(\lambda x_i + (1-\lambda)x_j) + \cdots \\ &\vdots \\ &= \lambda (y - f_\theta(x_i))^\intercal (y - f_\theta(x_i)) + (1-\lambda)(y - f_\theta(x_j))^\intercal (y - f_\theta(x_j))\end{aligned}$$

Therefore, one can readily show that:

$$\log p_\theta(y|x_{i,j,\lambda}) = Z + \lambda \log p_\theta(y|x_i) + (1-\lambda)\log p_\theta(y|x_j) \tag{96}$$

or equivalently that:

$$p_\theta(y|x_{i,j,\lambda}) \propto [p_\theta(y|x_i)]^\lambda [p_\theta(y|x_j)]^{1-\lambda} \tag{97}$$

Therefore, in this model setting, mixup and ProbMix (under log-linear pooling) are equivalent. □

Proof of Theorem 4.7. As noted in Section 3.2.2 and Appendix C, log-linear fusion of homoscedastic Gaussian embeddings results in a Gaussian distribution with the same variance σ and mean μ_θ equal to:

$$\mu_\theta(x_i, x_j) = \lambda h_\theta(x_i) + (1-\lambda)h_\theta(x_j).$$

If the mean is propagated during training and inference, this implies that the log-likelihood for M-ProbMix is:

$$\log p(y|x_{i,j,k}) = \log p\left(y|d_\phi(\lambda h_\theta(x_i) + (1-\lambda)h_\theta(x_j))\right),$$

which is equal to the manifold mixup log-likelihood.

□

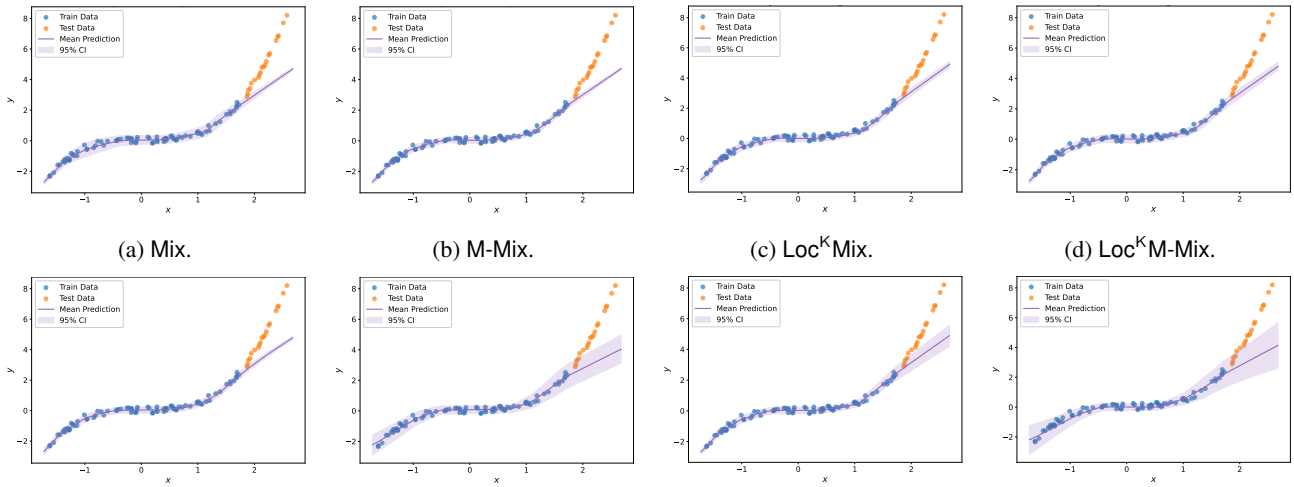
F EXPERIMENTAL RESULTS

F.1 TOY DATASETS

Here, we include additional results related to the toy regression and toy classification datasets.

F.1.1 Toy Regression Ablations

Figure 4 shows the a plot of the conditional density estimator obtain via each of the baseline methods (Mix, M-Mix, and Loc^{K}Mix, and Loc^{K}M-Mix) as compared to the proposed appraoches (ProbMix, M-ProbMix, and Loc^{K}ProbMix, and Loc^{K}M-ProbMix). We can see that methods like Mix and M-Mix actually attenuate uncertainty in the out-of-sample region, which can be problematic for risk-sensitive applications. ProbMix also performs poorly on out-of-sample data; we attribute this to the fact that fusing the log-likelihoods during training imposes a large bias for non-neighboring samples in a regression task. This flaw is resolved in all other variants of probabilistic mixup as the density plots in Fig. 4 show that M-ProbMix, Loc^{K}ProbMix, and Loc^{K}M-ProbMix perform best in terms of capturing uncertainty on out-of-sample data.

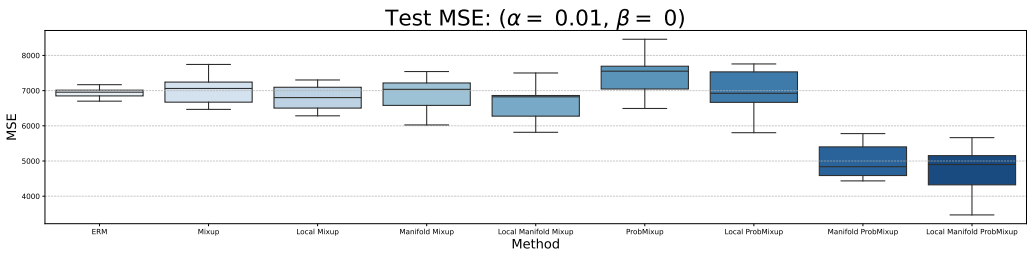


(a) Mix. (b) M-Mix. (c) Loc^{K}Mix. (d) Loc^{K}M-Mix.

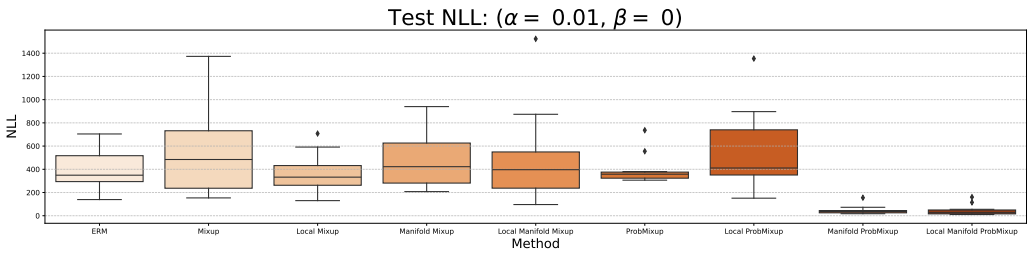
(e) ProbMix. (f) M-ProbMix. (g) Loc^{K}ProbMix. (h) Loc^{K}M-ProbMix.

Figure 4: Visual example of different approaches for toy regression. Loc^{K}ProbMix, and Loc^{K}M-ProbMix have widest uncertainty bounds on out-of-sample inputs, indicaitng that these variants are best calibrated in terms of uncertainty.

As an ablation study, we tested a grid of hyperparameter values defined by $\alpha \in \{0.01, 0.05, 0.10, 0.5, 1.0\}$ and $\beta \in \{0, 0.01\}$. We show a series of box plots comparing the MSE and the NLL across different methods. Results show that in the case of the regression experiment, manifold probabilistic mixup approaches (M-ProbMix and Loc^{K}M-ProbMix) achieve best performance in terms of both MSE and NLL. Please refer to Figures 5-14 for more details on the results.



(a) Average MSE.

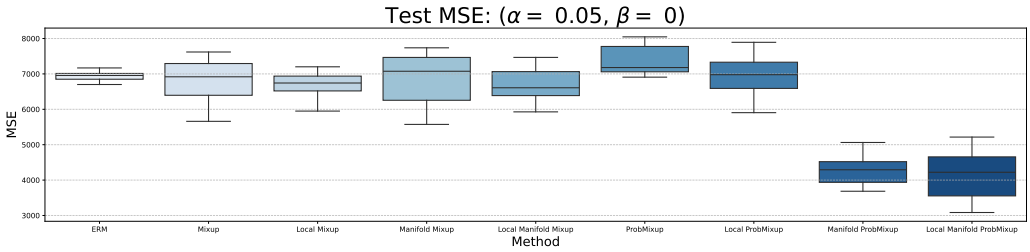


(b) Average negative log-likelihood.

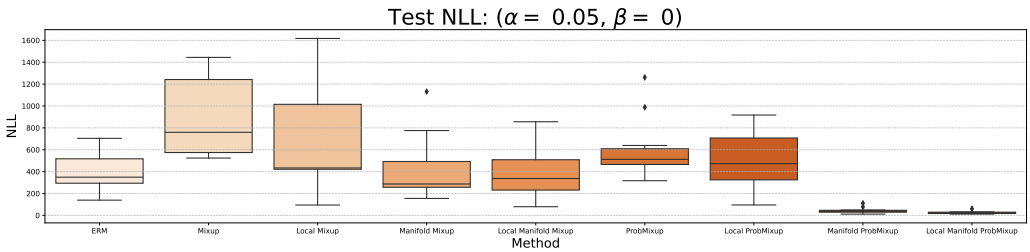
Figure 5: Toy regression results for $\alpha = 0.01$ and $\beta = 0$.

F.1.2 Toy Regression: Linear vs. Log-Linear Pooling

Here, we provide an additional ablation study comparing the performance of linear and log-linear pooling in ProbMix in the context of the Gaussian regression toy example. For linear pooling, the fusion functions g_{λ}^{x} and g_{λ}^{y} are assumed to be the

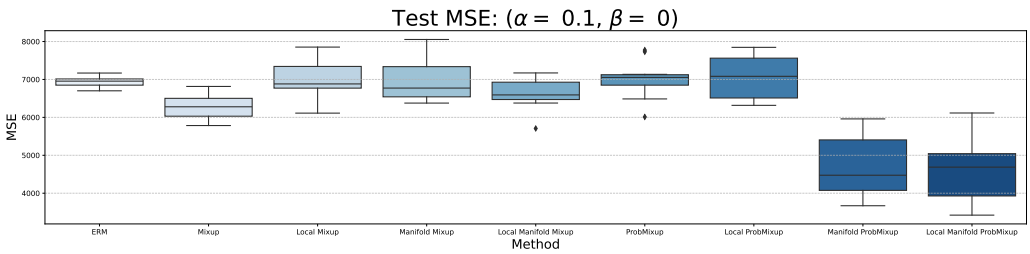


(a) Average MSE.

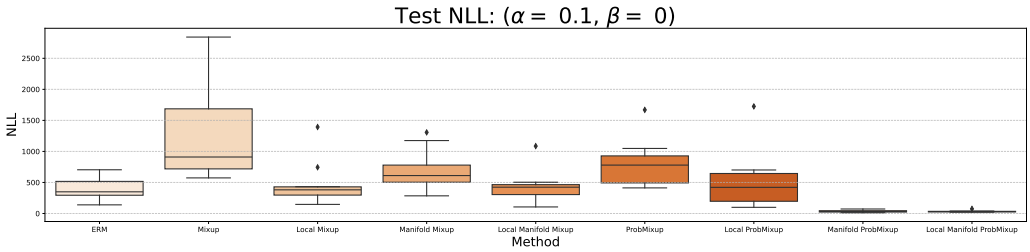


(b) Average negative log-likelihood.

Figure 6: Toy regression results for $\alpha = 0.05$ and $\beta = 0$.

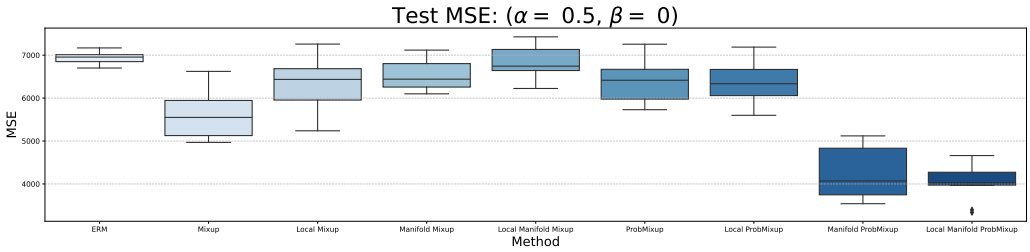


(a) Average MSE.

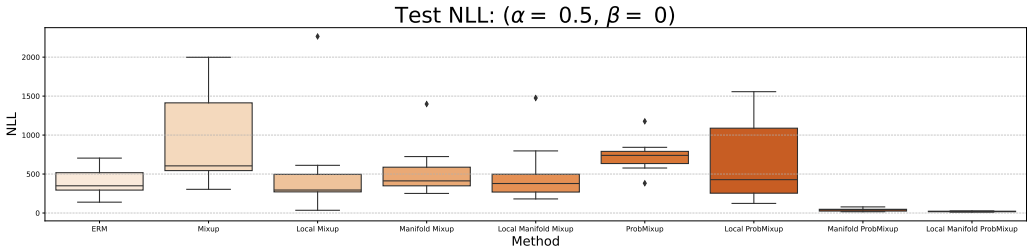


(b) Average negative log-likelihood.

Figure 7: Toy regression results for $\alpha = 0.1$ and $\beta = 0$.

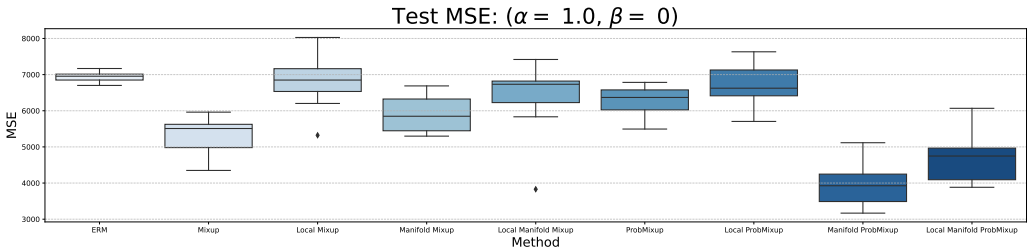


(a) Average MSE.

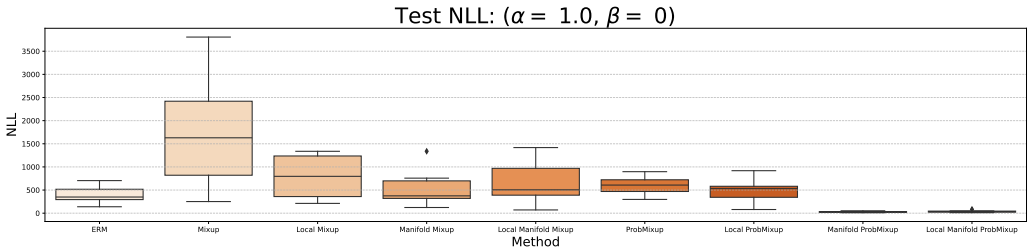


(b) Average negative log-likelihood.

Figure 8: Toy regression results for $\alpha = 0.5$ and $\beta = 0$.

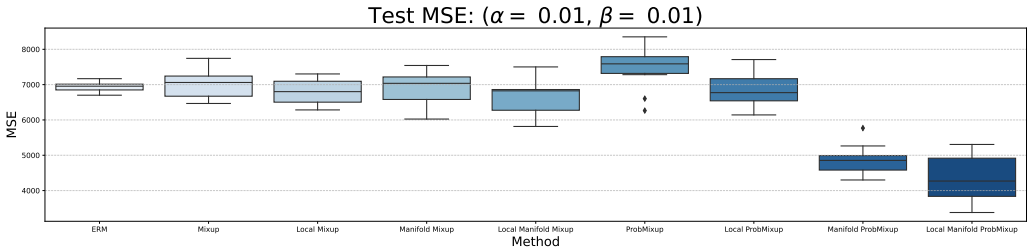


(a) Average MSE.

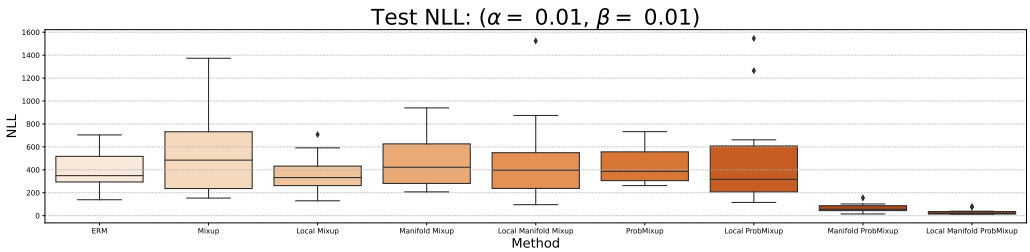


(b) Average negative log-likelihood.

Figure 9: Toy regression results for $\alpha = 1$ and $\beta = 0$.

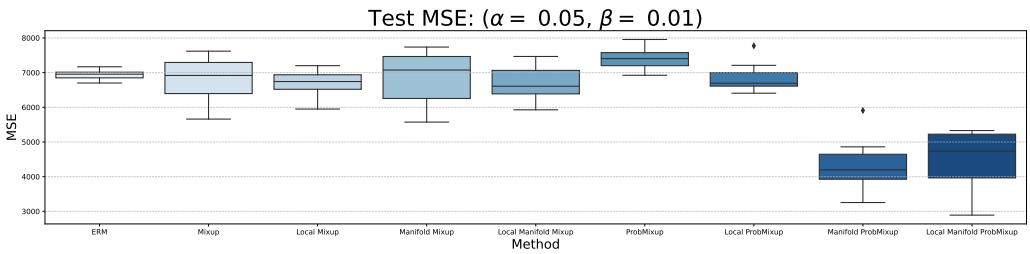


(a) Average MSE.

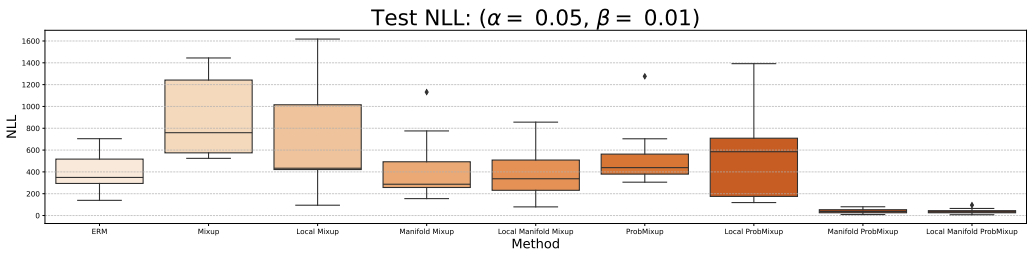


(b) Average negative log-likelihood.

Figure 10: Toy regression results for $\alpha = 0.01$ and $\beta = 0.01$.

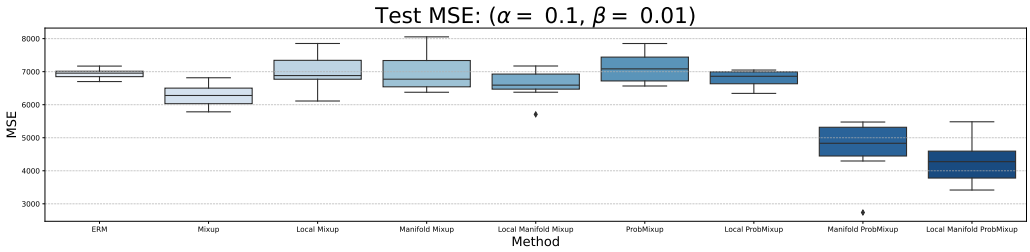


(a) Average MSE.

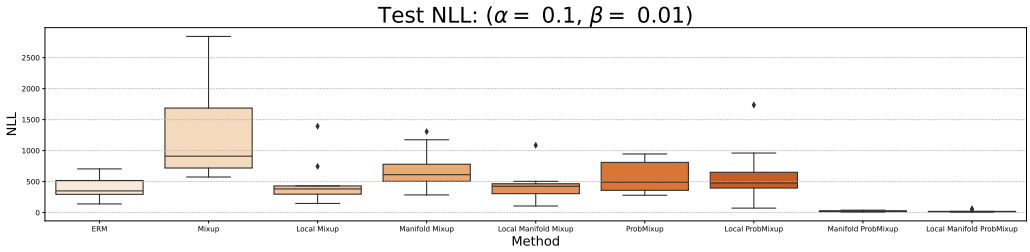


(b) Average negative log-likelihood.

Figure 11: Toy regression results for $\alpha = 0.05$ and $\beta = 0.01$.

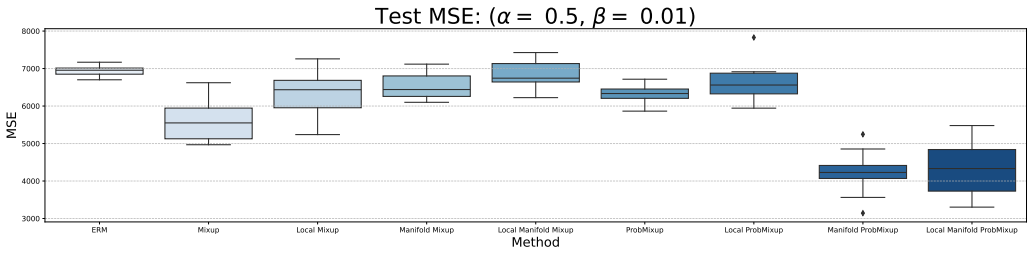


(a) Average MSE.

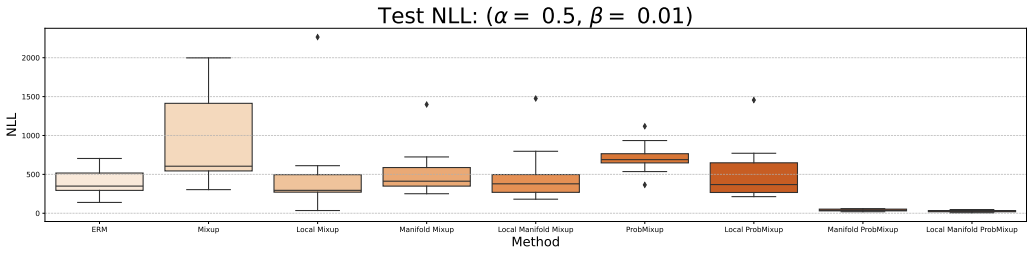


(b) Average negative log-likelihood.

Figure 12: Toy regression results for $\alpha = 0.1$ and $\beta = 0.01$.

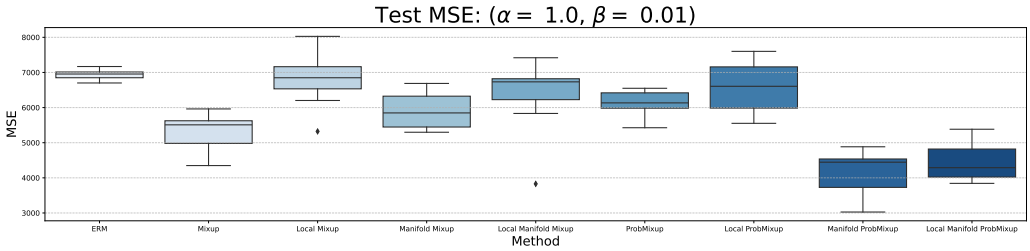


(a) Average MSE.

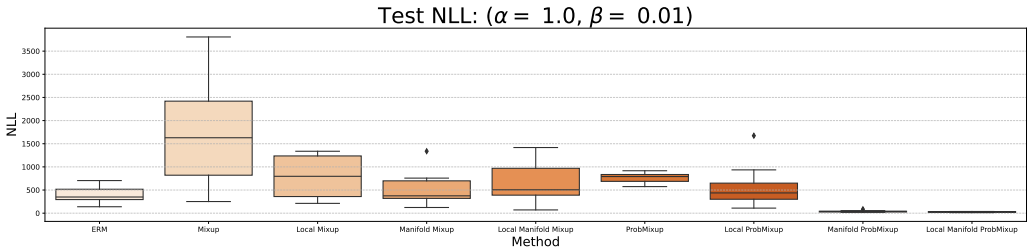


(b) Average negative log-likelihood.

Figure 13: Toy regression results for $\alpha = 0.5$ and $\beta = 0.01$.



(a) Average MSE.



(b) Average negative log-likelihood.

Figure 14: Toy regression results for $\alpha = 1.0$ and $\beta = 0.01$.

following:

$$\tilde{p}_\theta(y|x_i, x_j, \lambda) = g^x_\lambda(p_\theta(y|x_i), p_\theta(y|x_j)) = \lambda p_\theta(y|x_i) + (1-\lambda)p_\theta(y|x_j) \tag{98}$$

$$p_\beta(\tilde{y}|y_i, y_j, \lambda) = g^y_\lambda(s_\beta(\tilde{y}|y_i), s_\beta(\tilde{y}|y_j)) = \lambda s_\beta(\tilde{y}|y_i) + (1-\lambda)s_\beta(\tilde{y}|y_j) \tag{99}$$

We summarize the results in the ablation on the fusion function in terms of the average NLL for in distribution (ID) data and out-of-distribution (OOD) data in Tables 3 and 4, respectively. Based on the results, our findings are as follows:

1. For M-ProbMix and Loc^{K}M-ProbMix, linear pooling actually achieves better OOD performance than log-linear pooling. We can see in Table 4, for example with $\alpha = 0.5$, that M-ProbMix with linear pooling achieves an average NLL of 6.1, while the best performing ProbMix variant with log-linear pooling (Loc^{K}M-ProbMix) only achieves an average NLL of 16.9.
2. Improvement of the model in terms of OOD performance comes at a cost of ID performance. We can see that across all settings of α, M-ProbMix with linear pooling achieves much higher NLL than its log-linear counterpart (e.g., for $\alpha = 0.5$. 0.432 vs. -0.285) due to overcoverage of the quantified uncertainty.

α	ProbMix. (Linear)	Loc^{K}ProbMix. (Linear)	M-ProbMix. (Linear)	Loc^{K}M-ProbMix. (Linear)	ProbMix. (Log-Linear)	Loc^{K}ProbMix. (Log-Linear)	M-ProbMix. (Log-Linear)	Loc^{K}M-ProbMix. (Log-Linear)
$1e^{-32}$	-0.76±0.01	-0.66±0.02	-0.38±0.01	-0.30±0.01	**-0.78±0.01**	-0.62±0.02	-0.38±0.01	-0.28±0.03
0.01	**-0.77±0.01**	-0.66±0.01	-0.33±0.01	-0.30±0.02	-0.72±0.01	-0.67±0.01	-0.39±0.02	-0.33±0.01
0.05	**-0.76±0.01**	-0.62±0.02	-0.01±0.02	-0.30±0.01	-0.71±0.01	-0.65±0.01	-0.36±0.01	-0.29±0.01
0.10	**-0.74±0.01**	-0.66±0.02	0.11±0.04	-0.31±0.02	-0.66±0.01	-0.67±0.01	-0.35±0.01	-0.32±0.01
0.50	**-0.76±0.01**	-0.65±0.02	0.43±0.04	-0.29±0.02	-0.65±0.01	-0.64±0.01	-0.28±0.01	-0.30±0.01
1	**-0.74±0.01**	-0.66±0.02	0.36±0.03	-0.25±0.01	-0.51±0.02	-0.51±0.02	-0.25±0.01	-0.33±0.01

Table 3: Linear vs. Log-linear, changing hyper-parameter α, in-distribution NLL.

F.1.3 Toy Regression: Latent Variable Modeling

We further present results comparing mixup methods and ProbMix variants across different values of α. The goal of this ablation is to understand whether or not latent variable modeling (i.e., mapping to a statistical manifold in the latent space) is

α	ProbMix. (Linear)	Loc^{K}ProbMix. (Linear)	M-ProbMix. (Linear)	Loc^{K}M-ProbMix. (Linear)	ProbMix. (Log-Linear)	Loc^{K}ProbMix. (Log-Linear)	M-ProbMix. (Log-Linear)	Loc^{K}M-ProbMix. (Log-Linear)
$1e^{-32}$	94.9±11.8	157.1±40.8	21.3±3.3	**16.0±3.2**	92.6±12.9	229.4±133.6	29.5±6.6	78.6±62.5
0.01	103.5±15.0	171.1±41.0	28.4±8.8	22.7±4.3	78.1±8.1	132.7±38.7	27.5±6.2	**18.6±6.3**
0.1	90.2±12.7	180.7±30.0	**6.6±0.9**	17.4±3.0	117.8±27.0	157.2±18.1	31.0±5.4	17.0±2.2
0.1	85.3±13.5	206.9±30.6	**12.3±3.6**	24.4±6.4	173.1±27.3	258.4±52.7	25.9±6.1	32.8±8.1
0.5	83.4±10.17	149.2±35.5	**6.1±1.4**	22.0±3.9	313.6±59.6	176.2±55.0	47.7±12.2	16.9±3.0
1	91.8±7.1	104.5±22.9	**7.8±0.7**	14.8±4.8	370.2±54.3	117.4±18.6	37.2±13.5	23.8±5.6

Table 4: Linear vs. Log-linear, changing hyper-parameter α, OOD NLL.

the main cause of the strong performance of M-ProbMix and Loc^{K}M-ProbMix in the toy regression experiment. To that end, we fix $\beta = 10^{-32}$ and run all baselines across different values of $\alpha \in \{10^{-32}, 0.01, 0.05, 0.1, 0.25, 0.5, 1\}$. By varying the value of α, we can understand whether or not mixing is providing additional boost in performance on the OOD distribution NLL. We provide the OOD NLL for all baselines considered in Table 5. From the table, we observe that latent variable modeling alone ($\alpha = 10^{-32}$) provides a huge benefit on this toy experiment, improving the negative log-likelihood by nearly an order of magnitude (in comparison to mixup variants, ProbMix and Loc^{K}ProbMix). Interestingly enough, we can see that in addition to the benefits gained from latent variable modeling, performance can be further improved by incorporating ProbMix, as we get the best performance in the case of M-ProbMix when $\alpha = 0.25$ (22.9 ± 6.6) and Loc^{K}M-ProbMix when $\alpha = 0.5$ (16.9 ± 3.0).

	Mixup Methods				ProbMixup Methods			
α	Mix.	Loc^{K}Mix.	M-Mix.	Loc^{K}M-Mix.	ProbMix.	Loc^{K}ProbMix.	M-ProbMix.	Loc^{K}M-ProbMix.
$1e^{-32}$	**87.1±10.5**	143.2±31.7	**87.1±10.5**	**143.2±31.7**	92.6±13.0	229.4±133.6	29.5±6.5	78.6±62.4
0.01	98.6±24.2	**110.7±11.0**	162.5±47.9	152.3±45.4	**78.1±8.1**	132.7±38.7	27.5±6.2	18.6±6.3
0.05	176.3±45.5	167.1 ± 52.8	136.2±33.5	147.1±25.8	117.8±27.0	157.2±18.1	31.0±5.4	**16.9±2.2**
0.1	258.92±51.1	202.1±41.7	148.5 ±29.9	177.8±26.7	173.1±27.3	258.4±52.7	25.9 ±6.0	32.8±8.1
0.25	270.3±62.2	207.5±44.5	140.3±33.5	204.3±58.1	302.0±30.8	120.2±18.4	**22.9 ± 6.7**	19.1±3.9
0.5	306.0±38.9	219.1±57.7	136.4±23.8	202.4±34.3	313.6±59.6	176.2±55.0	47.7±12.2	**16.9 ± 3.0**
1	310.3±52.3	256.7±85.0	134.1±34.0	186.3±42.7	370.1±54.3	**117.4±18.6**	37.1±13.4	23.8±5.6

Table 5: Changing hyper-parameter α, latent variable modeling.

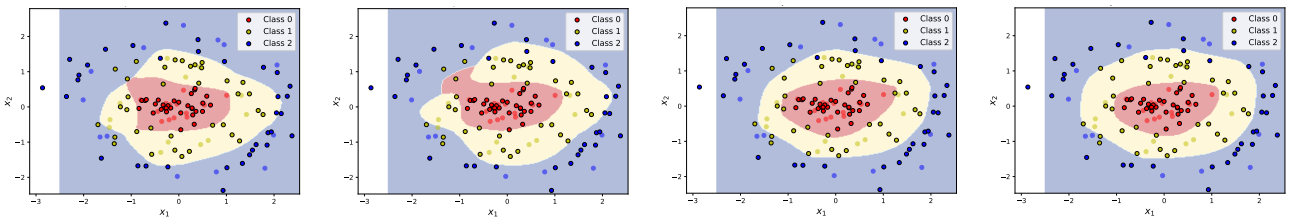
F.1.4 Toy Classification

Figure 15 shows the a plot of the decision boundaries of each network obtain via each of the baseline methods (Mix, M-Mix, and Loc^{K}Mix, and Loc^{K}M-Mix) as compared to the proposed approaches (ProbMix, M-ProbMix, and Loc^{K}ProbMix, and Loc^{K}M-ProbMix). Base on the plot of the decision boundaries, we can see that for this example ProbMix and Loc^{K}ProbMix yield the most reasonable decision boundaries, where Loc^{K}Mix and Loc^{K}M-Mix are close (but less noisy competitor). We can see that Mix and M-Mix suffer from the manifold intrusion issue, since the decision boundaries for the red and yellow class are mixed and deviate more from the ground truth data generating process. We observe a similar problem in the M-ProbMix and Loc^{K}M-ProbMix methods, showing that in the case of this dataset fusing at the logit level gives better generalization results than fusing on some embedding.

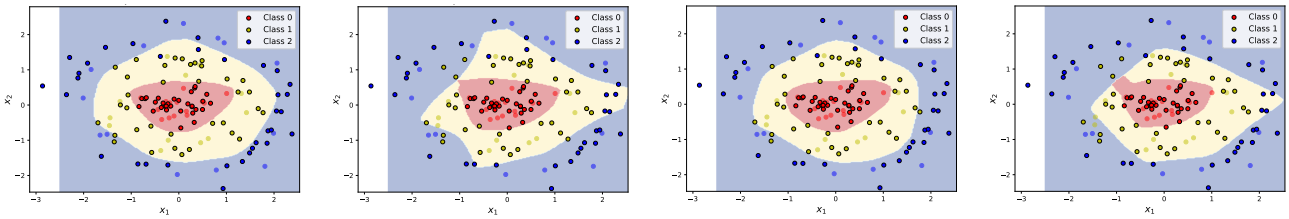
Similar to the case of the toy regression dataset, we conduct an ablation study for the α and β parameters for the toy classification dataset. We tested a grid of hyperparameter values defined by $\alpha \in \{0.01, 0.05, 0.10, 0.5, 1.0\}$ and $\beta \in \{0, 0.01\}$. We show a series of box plots comparing the test set accuracy and the NLL across different methods. Results show that in the case of the classification experiment, probabilistic mixup approaches without manifold augmentation (ProbMix and Loc^{K}ProbMix) achieve best performance in terms of both accuracy and NLL as compared to their manifold-based counterparts. Please refer to the results as shown in Figures 16-25 for more details on the results and the ablation studies

F.2 UCI

We include details of dataset and feature sizes below:



(a) Mix. (b) M-Mix. (c) Loc^{K}Mix. (d) Loc^{K}M-Mix.



(e) ProbMix. (f) M-ProbMix. (g) Loc^{K}ProbMix. (h) Loc^{K}M-ProbMix.

Figure 15: Visual example of different approaches for toy classification.

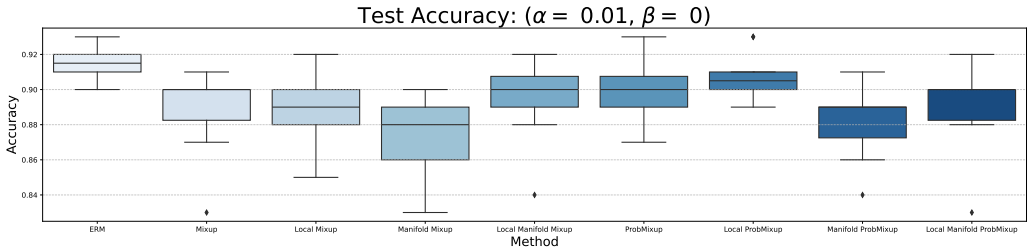
- *bostonHousing*: $n = 506, d = 13$;
- *concrete*: $n = 1030, d = 8$;
- *energy*: $n = 768, d = 8$;
- *kin8nm*: $n = 8192, d = 8$;
- *naval-propulsion-plant*: $n = 11934, d = 16$;
- *power-plant*: $n = 9568, d = 4$;
- *power-plant*: $n = 1599, d = 11$;
- *yacht*: $n = 308, d = 6$.

		Mixup Methods				ProbMixup Methods					
Dataset	ERM	Mix.	Loc^{K}Mix.	M-Mix.	Loc^{K}M-Mix.	ProbMix.	Loc^{K}ProbMix.	M-ProbMix.	Loc^{K}M-ProbMix.	M-ProbMix.*	Loc^{K}M-ProbMix.*
bostonHousing	5.77 ± 3.23	4.68 ± 3.00	5.25 ± 3.33	4.65 ± 2.67	4.72 ± 2.86	4.27 ± 2.75	5.00 ± 3.18	4.54 ± 2.84	3.93 ± 2.52	3.27 ± 0.97	$\mathbf{3.23 \pm 1.03}$
energy	$\mathbf{0.44 \pm 0.09}$	0.51 ± 0.10	0.45 ± 0.07	0.46 ± 0.08	0.45 ± 0.08	0.47 ± 0.09	0.48 ± 0.08	0.48 ± 0.07	0.49 ± 0.08	1.04 ± 0.46	0.76 ± 0.16
wine-quality-red	0.76 ± 0.10	0.72 ± 0.10	0.72 ± 0.10	0.77 ± 0.09	0.71 ± 0.09	0.75 ± 0.10	0.74 ± 0.10	0.77 ± 0.10	0.72 ± 0.10	$\mathbf{0.66 \pm 0.08}$	$\mathbf{0.66 \pm 0.08}$
concrete	5.88 ± 2.67	5.10 ± 0.58	5.19 ± 0.59	5.76 ± 2.73	5.73 ± 2.45	5.47 ± 2.76	$\mathbf{5.08 \pm 0.75}$	5.90 ± 2.50	5.88 ± 2.41	5.48 ± 0.67	5.43 ± 0.54
power-plant	3.91 ± 0.18	3.93 ± 0.17	3.96 ± 0.15	3.93 ± 0.17	3.97 ± 0.15	$\mathbf{3.85 \pm 0.14}$	3.96 ± 0.19	4.09 ± 0.22	4.13 ± 0.21	4.21 ± 0.20	4.24 ± 0.20
yacht	1.26 ± 2.71	1.96 ± 0.59	1.37 ± 0.37	0.70 ± 0.26	0.73 ± 0.30	0.92 ± 0.35	1.21 ± 0.58	$\mathbf{0.68 \pm 0.29}$	0.70 ± 0.31	1.41 ± 0.65	1.73 ± 0.74
kin8nm†	7.41 ± 0.34	7.73 ± 0.34	7.53 ± 0.37	$\mathbf{7.28 \pm 0.26}$	7.62 ± 0.42	7.52 ± 0.36	7.54 ± 0.42	7.66 ± 0.52	7.73 ± 0.35	8.01 ± 0.43	8.12 ± 0.51
naval-propulsion-plant†	0.21 ± 0.33	0.06 ± 0.01	0.09 ± 0.21	0.05 ± 0.03	$\mathbf{0.04 \pm 0.03}$	$\mathbf{0.04 \pm 0.07}$	0.09 ± 0.13	0.06 ± 0.02	0.17 ± 0.16	0.12 ± 0.01	0.13 ± 0.02

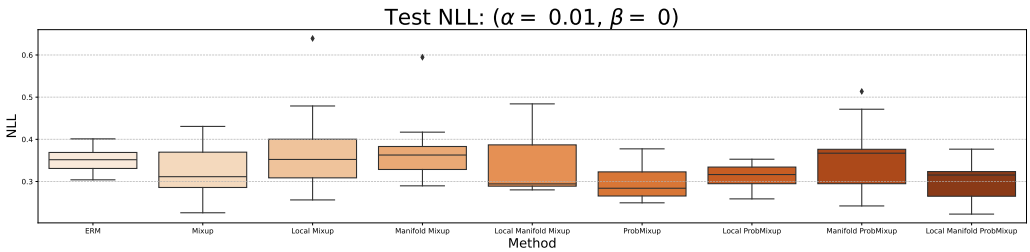
Table 6: RMSE for UCI regression datasets. †: normalized to a single integer digit. *: implements separate variance networks, for details see text. In a majority of the datasets, probabilistic mixup methods outperform ERM and mixup methods in terms of RMSE.

F.3 FINANCIAL FORECASTING

Here, we provide additional tables showing the performance of each method on the stock datasets in terms of average RMSE on the test set (please see Table 7). We also provide the average NLL and average RMSE on the training set (please see Tables 8 and 9). Finally, to understand the performance of the regularization techniques across the GME and NVDA stocks, we provide plots of the conditional density estimates corresponding to the best and worst NLL on both the training and test set. In particular, Figure 26 shows the conditional density estimates for the GME dataset using the LSTM predictor for the following regularization techniques: Mix, M-Mix, and Loc^{K}M-ProbMix. Figure 28 shows the conditional density estimates for the GME dataset using the Transformer predictor for the following regularization techniques: Mix and M-ProbMix. Finally, Figure 27 shows the conditional density estimates for the NVDA dataset using the LSTM predictor for the following techniques: ERM, ProbMix, and M-ProbMix. All in all, we find that methods that perform well in terms of NLL tend to

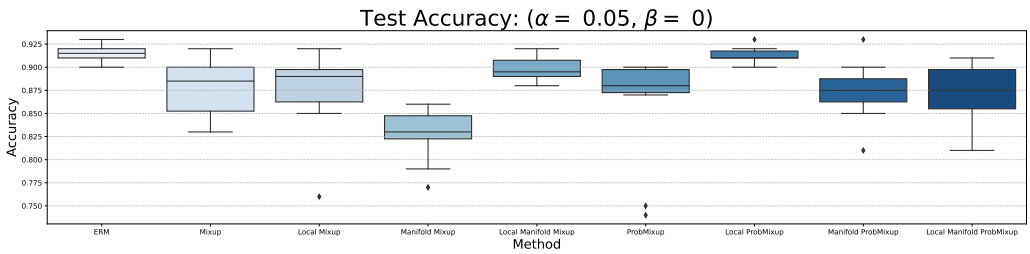


(a) Average accuracy.

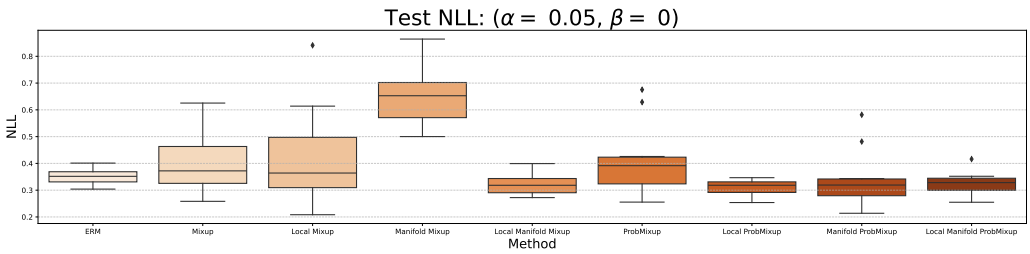


(b) Average negative log-likelihood.

Figure 16: Toy classification results for $\alpha = 0.01$ and $\beta = 0$.

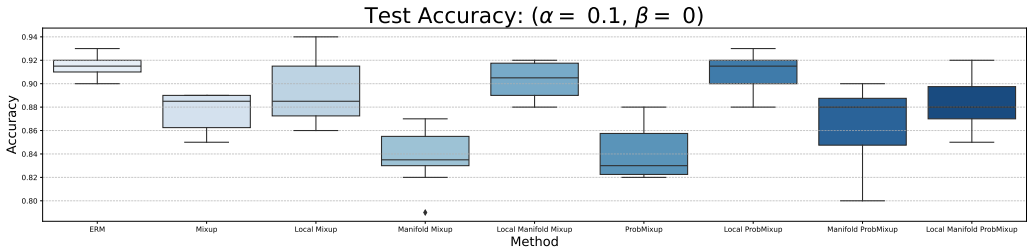


(a) Average accuracy.

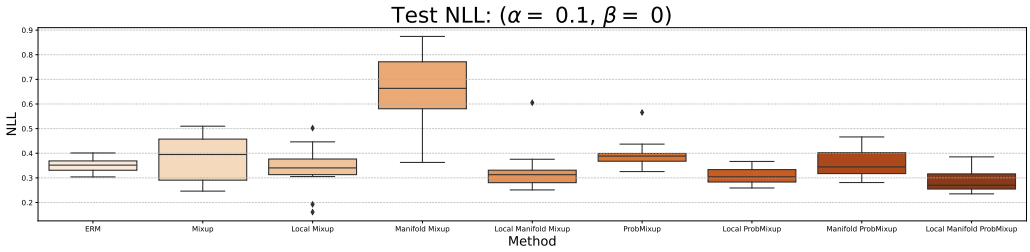


(b) Average negative log-likelihood.

Figure 17: Toy classification results for $\alpha = 0.05$ and $\beta = 0$.

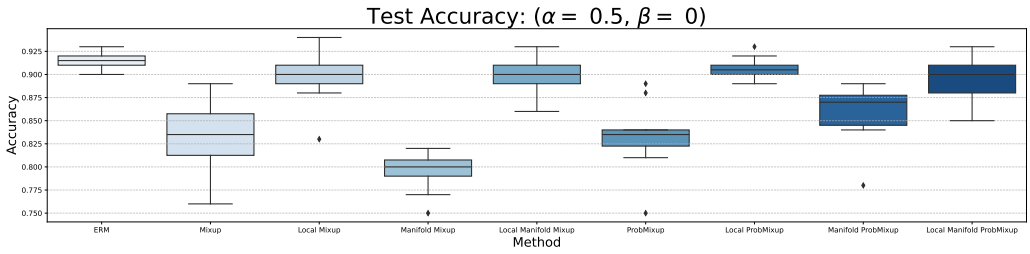


(a) Average accuracy.

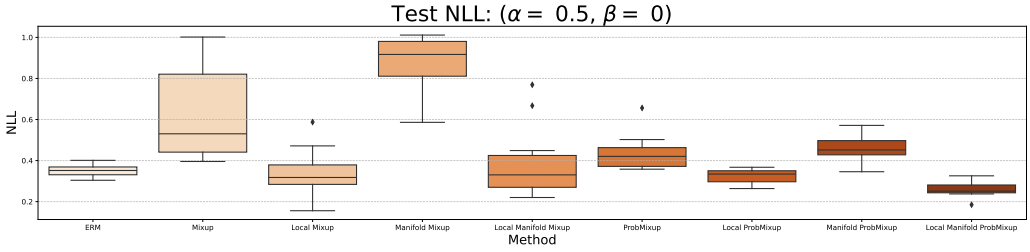


(b) Average negative log-likelihood.

Figure 18: Toy classification results for $\alpha = 0.1$ and $\beta = 0$.

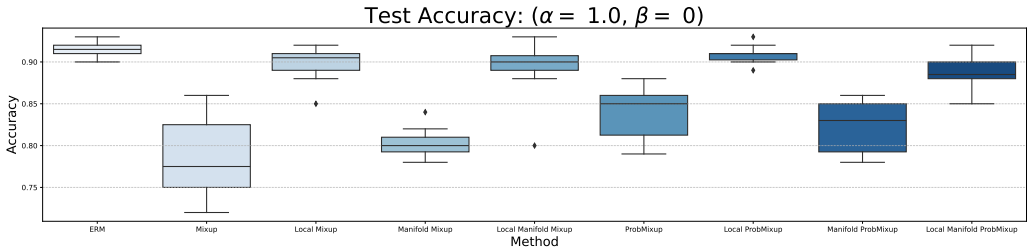


(a) Average accuracy.

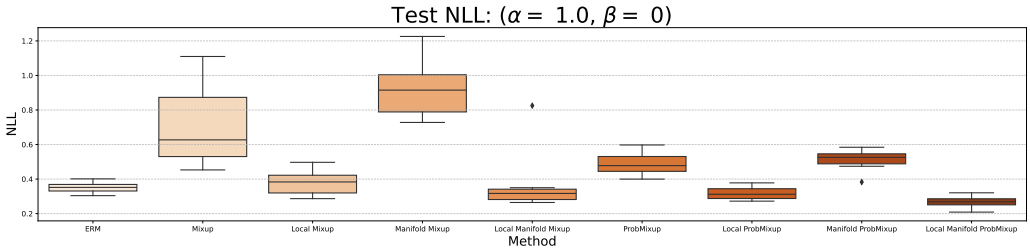


(b) Average negative log-likelihood.

Figure 19: Toy classification results for $\alpha = 0.5$ and $\beta = 0$.

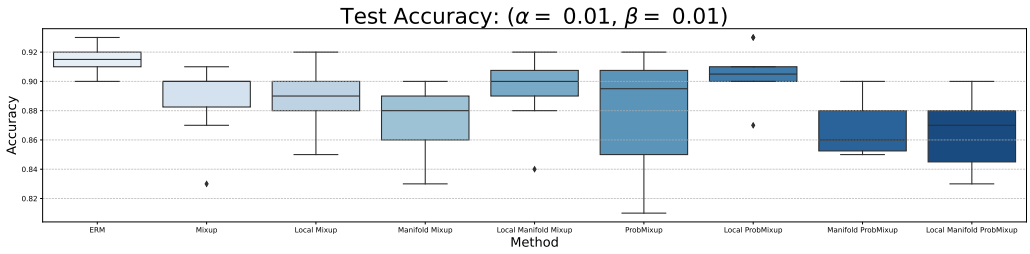


(a) Average accuracy.

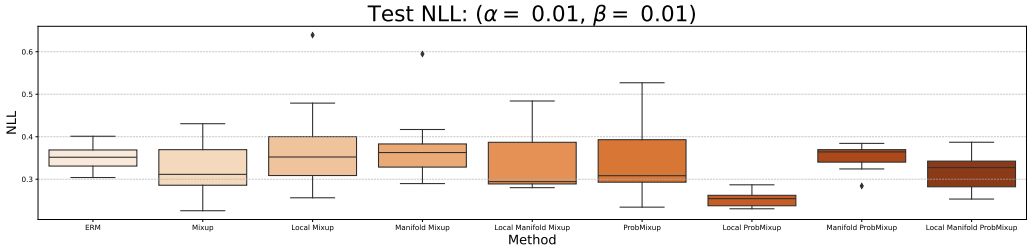


(b) Average negative log-likelihood.

Figure 20: Toy classification results for $\alpha = 1$ and $\beta = 0$.

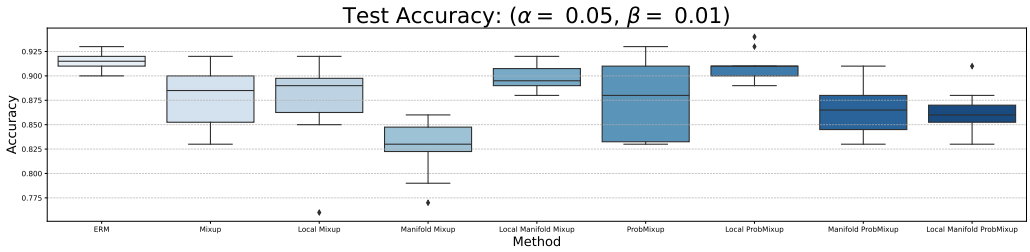


(a) Average accuracy.

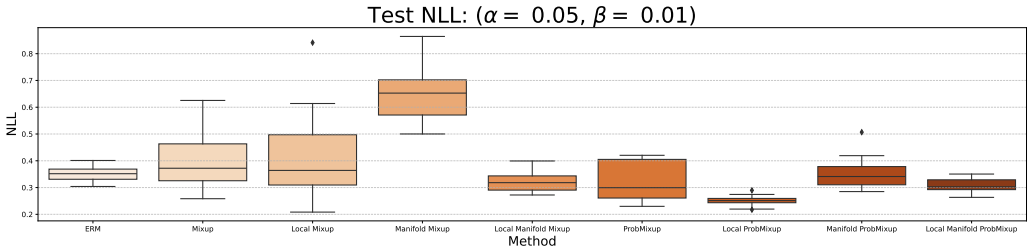


(b) Average negative log-likelihood.

Figure 21: Toy classification results for $\alpha = 0.01$ and $\beta = 0.01$.

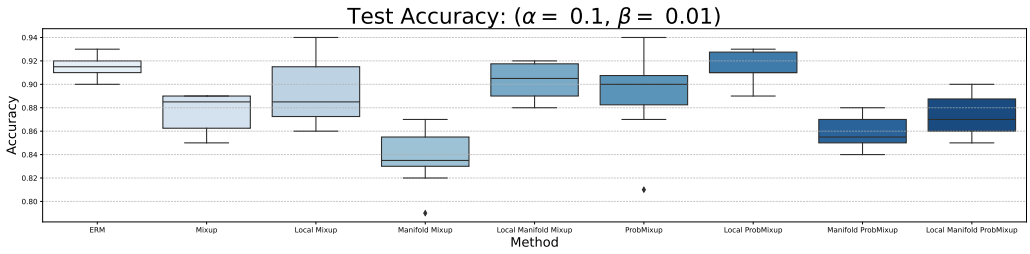


(a) Average accuracy.

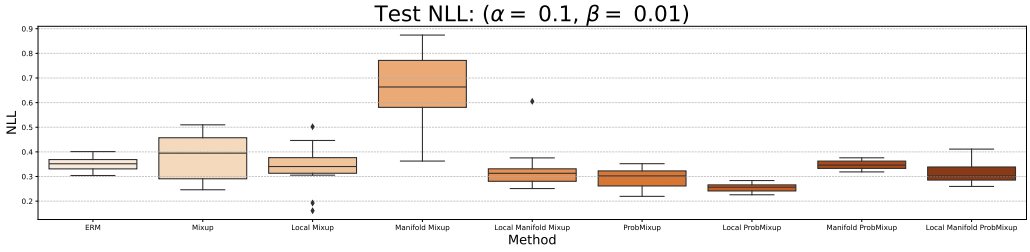


(b) Average negative log-likelihood.

Figure 22: Toy classification results for $\alpha = 0.05$ and $\beta = 0.01$.

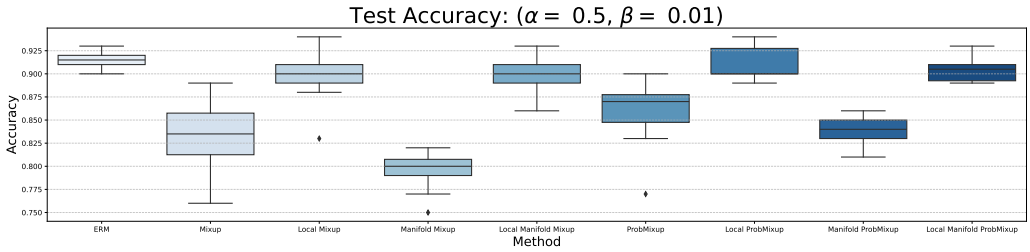


(a) Average accuracy.

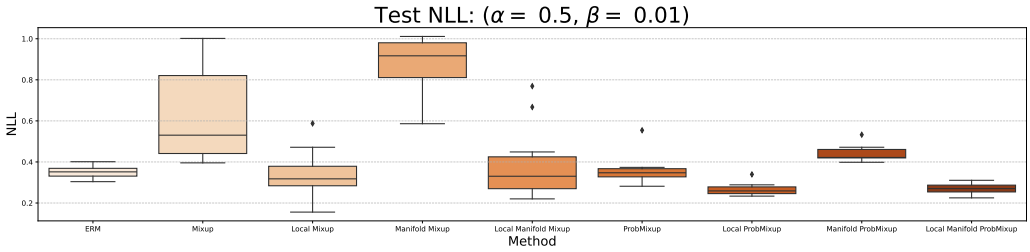


(b) Average negative log-likelihood.

Figure 23: Toy classification results for $\alpha = 0.1$ and $\beta = 0.01$.

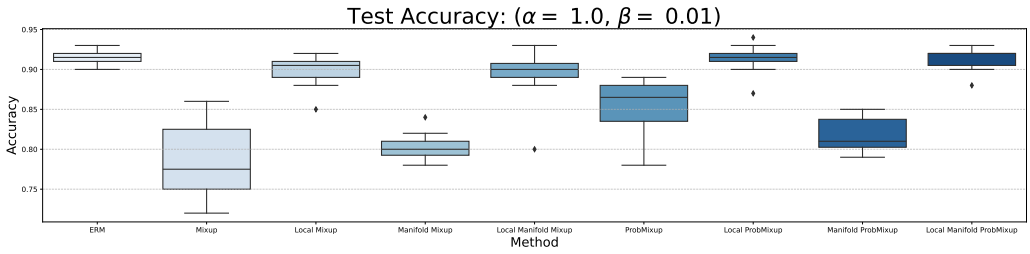


(a) Average accuracy.

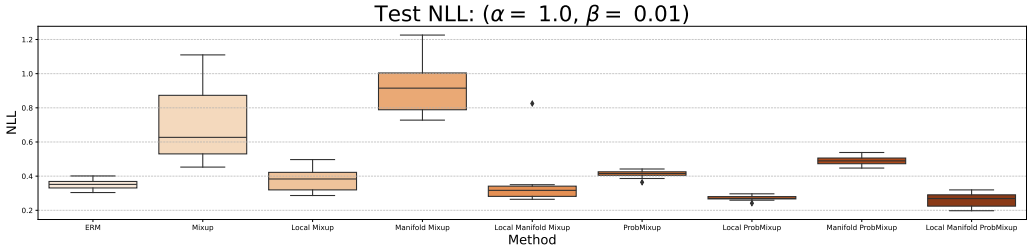


(b) Average negative log-likelihood.

Figure 24: Toy classification results for $\alpha = 0.5$ and $\beta = 0.01$.



(a) Average accuracy.



(b) Average negative log-likelihood.

Figure 25: Toy classification results for $\alpha = 1$ and $\beta = 0.01$.

have more conservative estimates of the variance of the conditional density, while models that performed poorly had both inaccurate means and variances.

LSTM		Mixup Methods				ProbMixup Methods			
Dataset	ERM	Mix	Loc^{K}Mix	M-Mix	Loc^{K}M-Mix	ProbMix	Loc^{K}ProbMix	M-ProbMix	Loc^{K}M-ProbMix
GME	1.64 ± 0.02	$\mathbf{1.48 \pm 0.02}$	1.60 ± 0.02	1.65 ± 0.01	1.62 ± 0.03	1.71 ± 0.03	1.65 ± 0.04	1.57 ± 0.01	1.57 ± 0.02
GOOG	2.06 ± 0.10	2.13 ± 0.05	1.87 ± 0.11	2.12 ± 0.09	1.98 ± 0.13	1.72 ± 0.16	$\mathbf{1.68 \pm 0.12}$	1.76 ± 0.10	1.81 ± 0.08
NVDA	0.25 ± 0.02	0.33 ± 0.04	0.42 ± 0.04	0.31 ± 0.03	0.44 ± 0.04	$\mathbf{0.24 \pm 0.02}$	0.39 ± 0.04	0.55 ± 0.02	0.40 ± 0.01
RCL	0.59 ± 0.02	$\mathbf{0.31 \pm 0.01}$	0.58 ± 0.02	0.65 ± 0.02	0.54 ± 0.01	0.62 ± 0.01	0.57 ± 0.02	0.49 ± 0.02	0.59 ± 0.02
Transformer		Mixup Methods				ProbMixup Methods			
Dataset	ERM	Mix	Loc^{K}Mix	M-Mix	Loc^{K}M-Mix	ProbMix	Loc^{K}ProbMix	M-ProbMix	Loc^{K}M-ProbMix
GME	1.82 ± 0.03	1.86 ± 0.01	1.85 ± 0.01	1.86 ± 0.01	1.79 ± 0.02	$\mathbf{1.78 \pm 0.02}$	1.84 ± 0.01	1.81 ± 0.01	1.82 ± 0.02
GOOG	2.24 ± 0.06	2.46 ± 0.03	$\mathbf{1.68 \pm 0.10}$	1.99 ± 0.09	2.06 ± 0.07	2.14 ± 0.10	1.79 ± 0.09	1.83 ± 0.13	2.00 ± 0.12
NVDA	0.36 ± 0.02	$\mathbf{0.28 \pm 0.01}$	0.42 ± 0.02	0.36 ± 0.01	0.35 ± 0.01	0.29 ± 0.01	0.36 ± 0.01	0.35 ± 0.02	0.37 ± 0.01
RCL	0.43 ± 0.01	$\mathbf{0.23 \pm 0.01}$	0.45 ± 0.01	0.42 ± 0.01	0.44 ± 0.01	0.42 ± 0.01	0.41 ± 0.01	0.40 ± 0.01	0.41 ± 0.01

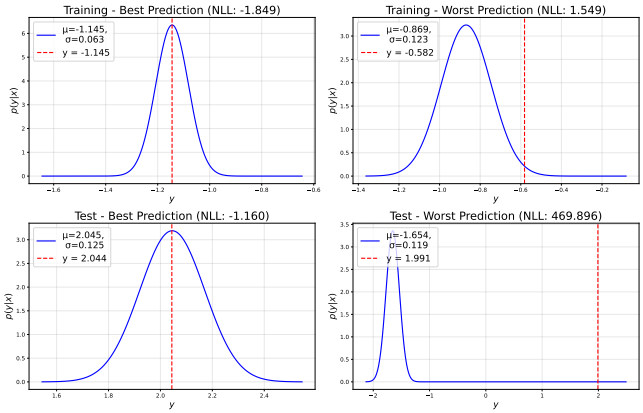
Table 7: RMSE for LSTM and Transformer models in the time series datasets. Results show that probabilistic mixup techniques do not lead to improvement in performance in terms of RMSE, as ERM and Mix outperform it on most of the datasets. This implies that while ProbMix may offer benefits in terms of calibrated uncertainty, its performance on uncertainty agnostic metrics like RMSE may not be up to par to ERM and other regularization schemes.

LSTM		Mixup Methods				ProbMixup Methods			
Dataset	ERM	Mix	Loc^{K}Mix	M-Mix	Loc^{K}M-Mix	ProbMix	Loc^{K}ProbMix	M-ProbMix	Loc^{K}M-ProbMix
GME	-1.39 ± 0.06	-1.25 ± 0.03	-1.74 ± 0.03	-1.42 ± 0.03	$\mathbf{-1.79 \pm 0.02}$	-1.46 ± 0.03	-1.70 ± 0.04	-0.94 ± 0.03	-1.26 ± 0.06
GOOG	-0.80 ± 0.06	-0.81 ± 0.02	$\mathbf{-1.19 \pm 0.03}$	-0.97 ± 0.04	-0.91 ± 0.04	-0.88 ± 0.04	-1.02 ± 0.03	-0.51 ± 0.03	-0.91 ± 0.04
NVDA	-1.56 ± 0.04	-1.23 ± 0.04	-1.76 ± 0.07	-1.57 ± 0.03	-1.84 ± 0.03	-1.50 ± 0.02	$\mathbf{-1.92 \pm 0.07}$	-0.94 ± 0.03	-1.30 ± 0.03
RCL	-1.29 ± 0.03	-0.98 ± 0.01	$\mathbf{-1.54 \pm 0.01}$	-1.31 ± 0.02	-1.47 ± 0.03	-1.11 ± 0.01	-1.52 ± 0.03	-0.56 ± 0.06	-1.08 ± 0.04
Transformer		Mixup Methods				ProbMixup Methods			
Dataset	ERM	Mix	Loc^{K}Mix	M-Mix	Loc^{K}M-Mix	ProbMix	Loc^{K}ProbMix	M-ProbMix	Loc^{K}M-ProbMix
GME	-1.41 ± 0.02	-1.28 ± 0.02	$\mathbf{-1.59 \pm 0.01}$	-1.44 ± 0.02	-1.51 ± 0.03	-1.34 ± 0.03	-1.52 ± 0.03	-1.42 ± 0.02	-1.49 ± 0.01
GOOG	-0.99 ± 0.04	-0.85 ± 0.02	-1.07 ± 0.02	-1.11 ± 0.02	-1.05 ± 0.01	-0.96 ± 0.03	-1.11 ± 0.01	-1.08 ± 0.01	$\mathbf{-1.15 \pm 0.01}$
NVDA	-1.62 ± 0.01	-1.52 ± 0.02	-1.72 ± 0.01	-1.47 ± 0.04	-1.69 ± 0.02	-1.48 ± 0.03	$\mathbf{-1.73 \pm 0.03}$	-1.49 ± 0.05	-1.71 ± 0.01
RCL	-1.24 ± 0.02	-0.88 ± 0.01	$\mathbf{-1.39 \pm 0.03}$	-1.20 ± 0.01	-1.31 ± 0.02	-1.00 ± 0.02	-1.32 ± 0.02	-1.19 ± 0.01	-1.26 ± 0.02

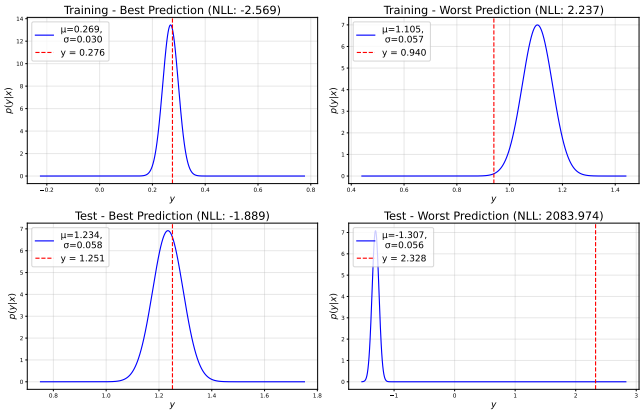
Table 8: NLL for LSTM and Transformer models in the time series datasets (in-sample performance).

LSTM		Mixup Methods				ProbMixup Methods			
Dataset	ERM	Mix	Loc^{K}Mix	M-Mix	Loc^{K}M-Mix	ProbMix	Loc^{K}ProbMix	M-ProbMix	Loc^{K}M-ProbMix
GME	0.09 ± 0.01	0.08 ± 0.00	0.07 ± 0.00	0.09 ± 0.01	$\mathbf{0.05 \pm 0.00}$	0.07 ± 0.00	0.06 ± 0.00	0.12 ± 0.00	0.08 ± 0.01
GOOG	0.18 ± 0.01	0.11 ± 0.00	$\mathbf{0.10 \pm 0.00}$	0.19 ± 0.02	0.13 ± 0.00	0.16 ± 0.02	0.11 ± 0.00	0.31 ± 0.02	0.22 ± 0.02
NVDA	0.07 ± 0.00	0.10 ± 0.01	0.06 ± 0.00	0.06 ± 0.00	$\mathbf{0.05 \pm 0.00}$	0.06 ± 0.00	$\mathbf{0.05 \pm 0.00}$	0.10 ± 0.00	0.08 ± 0.00
RCL	0.09 ± 0.00	0.09 ± 0.00	$\mathbf{0.06 \pm 0.00}$	0.08 ± 0.00	0.07 ± 0.00	0.09 ± 0.00	0.07 ± 0.00	0.15 ± 0.01	0.10 ± 0.00
Transformer		Mixup Methods				ProbMixup Methods			
Dataset	ERM	Mix	Loc^{K}Mix	M-Mix	Loc^{K}M-Mix	ProbMix	Loc^{K}ProbMix	M-ProbMix	Loc^{K}M-ProbMix
GME	0.08 ± 0.00	0.08 ± 0.00	$\mathbf{0.07 \pm 0.00}$	$\mathbf{0.07 \pm 0.00}$	$\mathbf{0.07 \pm 0.00}$	$\mathbf{0.07 \pm 0.00}$	$\mathbf{0.07 \pm 0.00}$	$\mathbf{0.07 \pm 0.00}$	$\mathbf{0.07 \pm 0.00}$
GOOG	0.15 ± 0.02	0.12 ± 0.00	0.14 ± 0.02	$\mathbf{0.10 \pm 0.00}$	0.16 ± 0.01	0.14 ± 0.02	0.12 ± 0.01	$\mathbf{0.10 \pm 0.00}$	0.11 ± 0.00
NVDA	0.06 ± 0.00	0.06 ± 0.00	$\mathbf{0.05 \pm 0.00}$	0.07 ± 0.00	$\mathbf{0.05 \pm 0.00}$	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	$\mathbf{0.05 \pm 0.00}$
RCL	0.10 ± 0.00	0.12 ± 0.00	$\mathbf{0.08 \pm 0.00}$	0.10 ± 0.00	0.09 ± 0.00	0.10 ± 0.00	0.09 ± 0.00	0.10 ± 0.00	0.10 ± 0.00

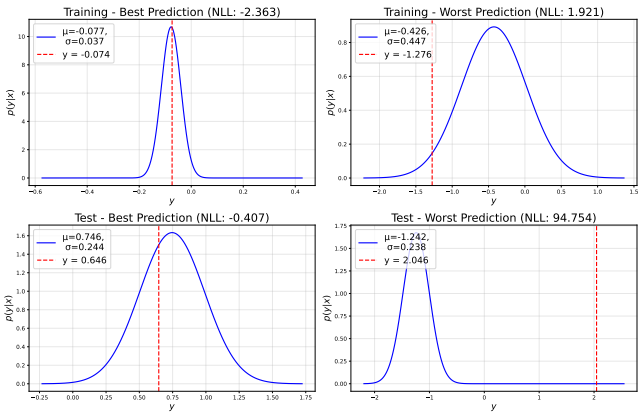
Table 9: RMSE for LSTM and Transformer models in the time series datasets (in-sample performance).



(a) Mix.

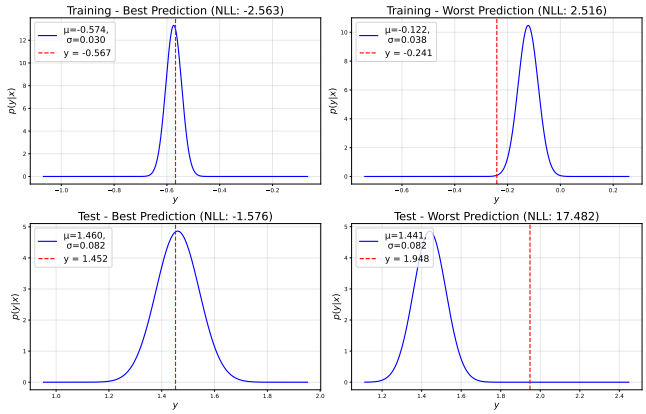


(b) M-Mix.

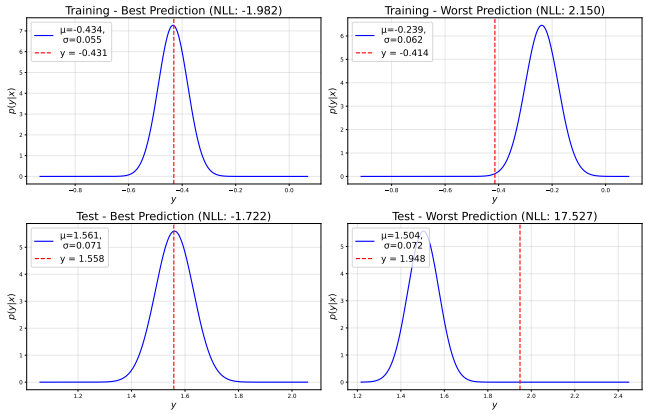


(c) Loc^{K}M-ProbMix.

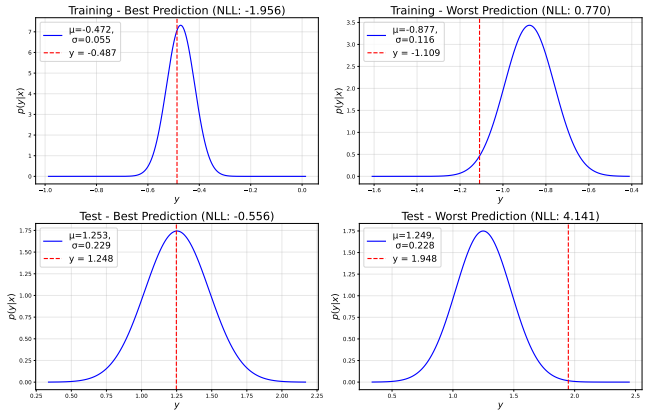
Figure 26: Predicted conditional densities with best and worst NLL values for LSTM model on GME dataset



(a) ERM.

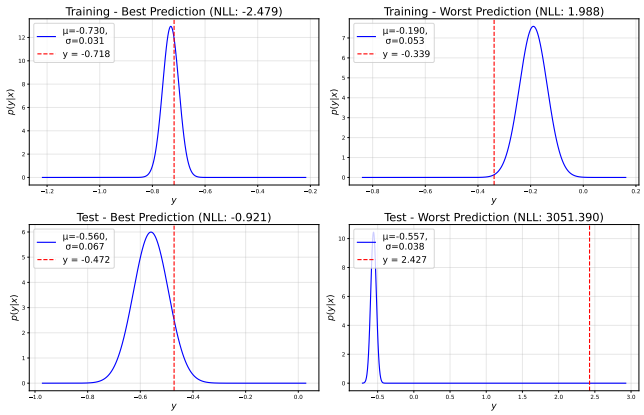


(b) ProbMix.

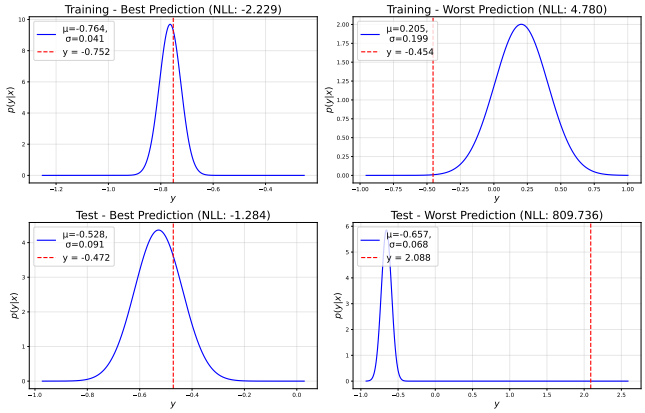


(c) Manifold probabilistic mixup.

Figure 27: Predicted conditional densities with best and worst NLL values for LSTM model on NVDA dataset



(a) Mix.



(b) M-ProbMix.

Figure 28: Predicted conditional densities with best and worst NLL values for Transformer model on GME dataset