# Deep Gaussian Mixture Ensembles (Supplementary Material)

**Yousef El-Laham**[1] **Niccolò Dalmasso**[1] **Elizabeth Fons**[1] **Svitlana Vyetrenko**[1]

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

## A THEORETICAL PROOFS

This section includes the proofs of the propositions presented in Section 4.3. We have also included the proposition statements for readability purposes.

**Proposition A.1.** *Under the assumption that $\pi_i = 1/K - 1$ for $i = 1, .., K$, maximizing the Gaussian mixture data likelihood directly achieves better or equal joint likelihood than maximizing each ensemble member's likelihood separately.*

*Proof.* The EM algorithm minimizes the joint data log-likelihood as defined in equation (**??**), which can be lower-bounded in the following way by using Jensen's inequality:

$$\arg\max_\theta \mathbb{E}_{X,Y}\left[\log\left(\sum_{k=1}^K \pi_k p_k(y|x,\theta_k)\right)\right] \geq \arg\max_\theta \mathbb{E}_{X,Y}\left[\sum_{k=1}^K \log(\pi_k) + \log(p_k(y|x,\theta_k))\right] =$$
$$= \arg\max_\theta \sum_{k=1}^K \mathbb{E}_{X,Y}[\log(\pi_k)] + \mathbb{E}_{X,Y}[\ell_{\theta_k}(x,y))] .$$

By assumption, the first term constant (of value $-\log(K)$), hence:

$$\arg\max_\theta \mathbb{E}_{X,Y}\left[\log\left(\sum_{k=1}^K \pi_k p_k(y|x,\theta_k)\right)\right] \geq \arg\max_\theta \sum_{k=1}^K \mathbb{E}_{X,Y}[\ell_{\theta_k}(x,y))] ,$$

with the lower bound corresponding to maximizing the likelihood of each ensemble member separately, as performed in DEs [Lakshminarayanan et al., 2017].

□

**Proposition A.2.** *Under assumptions 4.4 - 4.7, let the mean and variance in each ensemble model being estimated via a separate 2-layer deep ReLu network from a common feature extraction layer. Then the DGMEs EM algorithm convergences to a non stationary point that maximizes the data likelihood with high-probability.*

*Proof.* Using Wu [1983, Theorem 4.1], to guarantee convergence of the EM algorithm it is enough to prove that at every round $t$:

$$\forall \theta \notin \mathcal{N} : Q(\theta^{(t+1)}; \theta^{(t)}) - Q(\theta^{(t)}; \theta^{(t)}) > 0, \tag{1}$$

where $\mathcal{N}$ is the set of stationary points of the function $Q$. By writing the difference in equation (1) above we have that:

$$\begin{aligned} Q(\theta^{(t+1)}; \theta^{(t)}) - Q(\theta^{(t)}; \theta^{(t)}) &= \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{k,n} (\log(\pi_k) + \ell_{\theta^{(t+1)}}(x_n, y_n)) - \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{k,n} (\log(\pi_k) + \ell_{\theta^{(t)}}(x_n, y_n)) \\ &= \sum_{k=1}^{K} \left[ \sum_{n=1}^{N} \gamma_{k,n} \left( \ell_{\theta^{(t+1)}}(x_n, y_n) \right) - \sum_{n=1}^{N} \gamma_{k,n} \left( \ell_{\theta^{(t)}}(x_n, y_n) \right) \right]. \end{aligned}$$

By setting $\theta^{(t+1)} = \theta_k^*$ and using Assumption 4.4:

$$Q(\theta_k^*; \theta^{(t)}) - Q(\theta^{(t)}; \theta^{(t)}) \geq \sum_{k=1}^{K} \frac{\epsilon_{t,k}}{K} > \epsilon$$

The result follows if every ensemble network can learn the maximum likelihood $\theta^*$ at every round. We will show that the above happens in high probability. Without loss of generality, set the round $t$ and the ensemble member $k$ if the mean and variance functions follow assumptions 4.5 and 4.6. Let $\ell^* = \ell_{\theta_k^*}$ and $\hat{\ell}_\theta$ be the estimated likelihood. As the likelihood is Gaussian, the estimation problem is equivalent to estimating the true mean function $\mu^*(x)$ and variance function $\sigma^*(x)$. Assume the mean and variance functions are learnt independently by using a pre-trained feature extraction layer, we can break down the estimation problem into:

$$\begin{aligned} \|\ell(\mu^*, \sigma^*) - \ell(\widehat{\mu}, \widehat{\sigma})\|_2 &= \|\ell(\mu^*, \sigma^*) \pm \ell(\mu^*, \hat{\sigma}) - \ell(\widehat{\mu}, \widehat{\sigma})\|_2 \\ &\leq \underbrace{\|\ell(\mu^*, \widehat{\sigma}) - \ell(\widehat{\mu}, \widehat{\sigma})\|_2}_{(A)} + \underbrace{\|\ell(\mu^*, \sigma^*) - \ell(\mu^*, \widehat{\sigma})\|_2}_{(B)}. \end{aligned}$$

Provided $n > \mathcal{O}(\log(1/\delta)/\epsilon^2)$ and using Assumption 4.7 to guarantee non-degenerate weights, the proposition follows since:

(A) For the mean function estimation, the likelihood reduces to a weighted least square loss, which satisfies the assumptions in Farrell et al. [2021, 2.1]. Hence, one would need at least $n > \mathcal{O}(\log(1/\delta)/\epsilon)$ samples to estimate the mean function within $\epsilon/2$ radius and with probability $1 - \delta$;

(B) For the variance function estimation, the assumption correspond to the requirement in Arora et al. [2019, Section 5]; hence, one would need at least $n > \mathcal{O}(\log(1/\delta)/\epsilon^2)$ samples to estimate the mean function within an $\epsilon/2$ radius and with probability $1 - \delta$.

□

**Proposition A.3.** *If the weights of each ensemble members are initialized to 0 with fixed bias terms, a single EM step for DGMEs is equivalent to perform DEs.*

*Proof.* If any ensemble members $f_k$ has all weights initialized to 0, then it follows that $p_k(y_n|x_n, \theta_k) = a$ for some constant $\delta \in \mathbb{R}$. In addition, $\mu_{\theta_k}(x_n) = \mu$ and $\sigma^2_{\theta_k}(x_n)$ for any $x_n$. Hence, in the expectation steps all posterior probabilities are equal to:

$$\gamma_{k,n} = \frac{p_k(y_n|x_n, \theta_k) P_\theta(z_n = k)}{\sum_{j=1}^{K} p_j(y_n|x_n, \theta_j) P_\theta(z_n = j)} = \frac{\delta \mathcal{N}(y_n; \mu, \sigma^2)}{\sum_{j=1}^{K} \delta \mathcal{N}(y_n; \mu, \sigma^2)} = \frac{1}{K}.$$

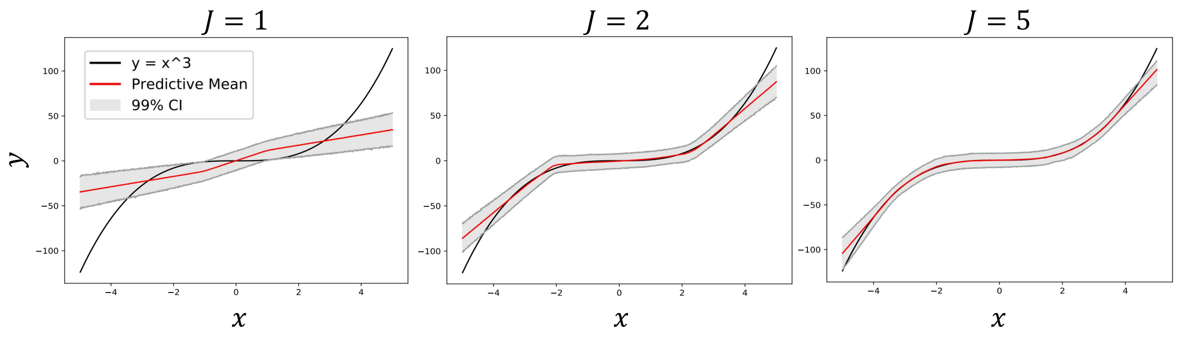

Figure 1: Results on a toy regression task with Gaussian noise for different numbers of EM rounds, as described in Section **??**. As $J$ increases, the predictive mean improves.

| | $(E = 1, J = 50)$ | $(E = 2, J = 25)$ | $(E = 5, J = 10)$ | $(E = 10, J = 5)$ | $(E = 25, J = 2)$ | $(E = 50, J = 1)$ |
|---|---|---|---|---|---|---|
| Normal - Unimodal | $2.71 \pm 0.06$ | $2.63 \pm 0.06$ | $2.58 \pm 0.03$ | $2.54 \pm 0.01$ | $2.54 \pm 0.01$ | $2.56 \pm 0.03$ |
| Heavy-Tailed - Unimodal | $2.98 \pm 0.03$ | $2.95 \pm 0.02$ | $2.88 \pm 0.02$ | $2.87 \pm 0.01$ | $2.91 \pm 0.01$ | $2.96 \pm 0.02$ |
| Normal - Bimodal | $3.15 \pm 0.05$ | $3.09 \pm 0.07$ | $3.02 \pm 0.04$ | $3.13 \pm 0.08$ | $3.42 \pm 0.06$ | $3.53 \pm 0.04$ |

Table 1: Training NLL obtained for the toy regression dataset using DGMEs under different configurations of the number of epochs per EM round $E$ and the total number of EM rounds $J$ for a fixed computational budget $E \times J = 50$.

Hence, the maximization in the M-step is equal to:

$$\theta_k^\star = \arg\max_{\theta_k \in \Theta_k} \sum_{n=1}^{N} \gamma_{k,n} \ell_{\theta_k}(x_n, y_n) = \arg\max_{\theta_k \in \Theta_k} \sum_{n=1}^{N} \ell_{\theta_k}(x_n, y_n),$$

which corresponds to maximizing the likelihood of each ensemble member separately, as performed in DE Lakshminarayanan et al. [2017].

□

## B ADDITIONAL EXPERIMENTAL RESULTS AND ABLATION STUDIES

### B.1 TOY REGRESSION

In this subsection, we provide additional experimental results and ablation studies on the toy regression dataset that provide valuable insights on the role of each of the DGME hyperparameters.

#### B.1.1 Ablation: Number of EM Rounds

To study the effect that the number of EM rounds has on training of DGMEs, Figure 1 shows DGMEs trained with 1, 2 and 5 rounds on the toy regression task with Gaussian noise (Case 1), where the number of epochs per round is fixed to $E = 80$. We can see in this figure that after $J = 5$ EM rounds, the algorithm has converged to a conditional distribution that represents the ground truth quite well.

Additionally, we can assess the joint impact of the number of epochs $E$ used in the M-Step per EM round and the total number of EM rounds $J$, while keeping the total computational budget constant (e.g., $E \times J = 50$ total epochs). We test the following values of $E \in \{1, 2, 5, 10, 25, 50\}$ and report the average NLL over the training set and its corresponding standard error (computed over a total of 10 runs) in Table 1.

We can see empirically that for a fixed computational budget, there is tradeoff between the performance and the effective number of EM rounds. The tradeoff is more apparent when considering the more difficult examples (i.e., heavy-tailed unimodal noise and normal bimodal noise). If the number of epochs in the M-step is $E = 1$ and we train for $J = 50$

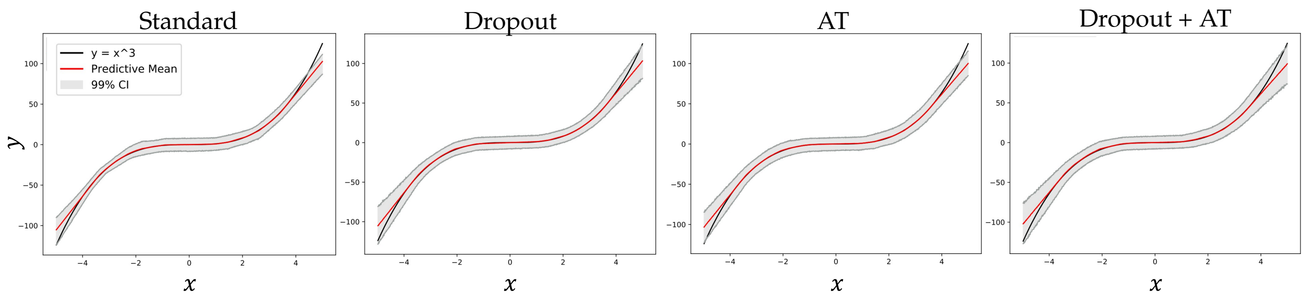

Figure 2: Results on a toy regression task with Gaussian noise. Left most plot corresponds to standard set up of DGMEs trained with $K = 5$ networks. Second plot corresponds to incorporating Dropout in the training. Third plot shows the effect of using adversarial training, and final plot shows the effect of using both dropout and adversarial training.

| | $p_d = 0.0$ | | $p_d = 0.05$ | | $p_d = 0.1$ | | $p_d = 0.15$ | | $p_d = 0.2$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Train NLL | Test NLL | Train NLL | Test NLL | Train NLL | Test NLL | Train NLL | Test NLL | Train NLL | Test NLL |
| Normal - Unimodal | $2.55 \pm 0.01$ | $7.50 \pm 0.88$ | $2.59 \pm 0.01$ | $4.86 \pm 0.28$ | $2.63 \pm 0.01$ | $4.30 \pm 0.12$ | $2.66 \pm 0.01$ | $4.17 \pm 0.08$ | $2.70 \pm 0.02$ | $4.10 \pm 0.06$ |
| Heavy-Tailed - Unimodal | $2.87 \pm 0.01$ | $6.31 \pm 0.49$ | $2.90 \pm 0.01$ | $4.83 \pm 0.17$ | $2.93 \pm 0.01$ | $4.44 \pm 0.15$ | $2.97 \pm 0.02$ | $4.23 \pm 0.08$ | $2.99 \pm 0.02$ | $4.19 \pm 0.06$ |
| Normal - Bimodal | $3.16 \pm 0.09$ | $6.81 \pm 1.08$ | $3.18 \pm 0.08$ | $5.80 \pm 0.37$ | $3.29 \pm 0.06$ | $5.44 \pm 0.24$ | $3.31 \pm 0.07$ | $5.32 \pm 0.20$ | $3.36 \pm 0.05$ | $5.34 \pm 0.09$ |

Table 2: Train and test NLL of DGMEs for each toy regression dataset under different dropout probability values.

rounds, not enough information is being propagated between the E- and M-Step in each round of training, making learning inefficient. In the other extreme, if the number of epochs per M-step is $E = 50$ and we train for $J = 1$ rounds, even if the optimization problem in the M-step is more accurately resolved, we do not run enough EM rounds to accurately learn the underlying conditional density function. If we balance the number of epochs per rounds and the total number of EM rounds (i.e., $(E = 5, J = 10)$ or $(E = 10, J = 5)$), we get much better performance in terms of training NLL.

### B.1.2 Ablation: Dropout and Adversarial Training

In this ablation, our goal is to understand the effect of epistemic uncertainty estimation techniques in DGMEs. As a rough analysis, Figure 2 shows the effect of training with dropout, adversarial training and their combination. Here, the dropout probability is set to $p_d = 0.05$. We can see that without dropout or adversarial training, the uncertainty estimates are well-calibrated for the training data (features taking value between -4 and 4), but are underestimated for the test data (features taking absolute value between 4 and 5). By incorporating dropout and adversarial training, we can see that the uncertainty estimates become larger for the test examples.

To get a better understanding of the effect of dropout probability $p_d$ on the quantified uncertainty, we can evaluate the train and test NLL for different values of $p_d$ for each of the toy datasets. Results are shown in Table 2. From this table, we observe that dropout creates a trade-off between performance on in-sample data and out-of-sample data in terms of NLL. Increasing the dropout probability in this case causes the average NLL to be worse for the training set, but improves it (up to a certain point) on the test set. In practice, we can choose the dropout probability to minimize the NLL on a validation set.

### B.1.3 Ablation: Number of Mixture Components

The number of components in the assumed Gaussian mixture impacts how well the model can estimate more complex noise distributions (e.g., heavy-tailed or bimodal distributions). Gaussian mixtures (with infinite components) are universal approximators to smooth continuous density functions, so the more components assumed, the more flexible the model is. When choosing the number of mixture components, one should take into consideration the complexity of the data generating process and the amount of data in the training set. If the data generating process is known to be Gaussian, then choosing a large number of components is not beneficial. On the other hand, if the data generating process is thought to be multimodal, then using more components is the better choice. We can see this in the following two ablation studies.

Figure 3 shows the effect of the number of mixtures components $K$ on the kurtosis of the learned predictive distribution. We observe that with more mixture components, the the model learns a fatter-tailed distribution. This makes sense since a Student-t distribution can be viewed as a Gaussian mixture with an infinite number of components with different variances.

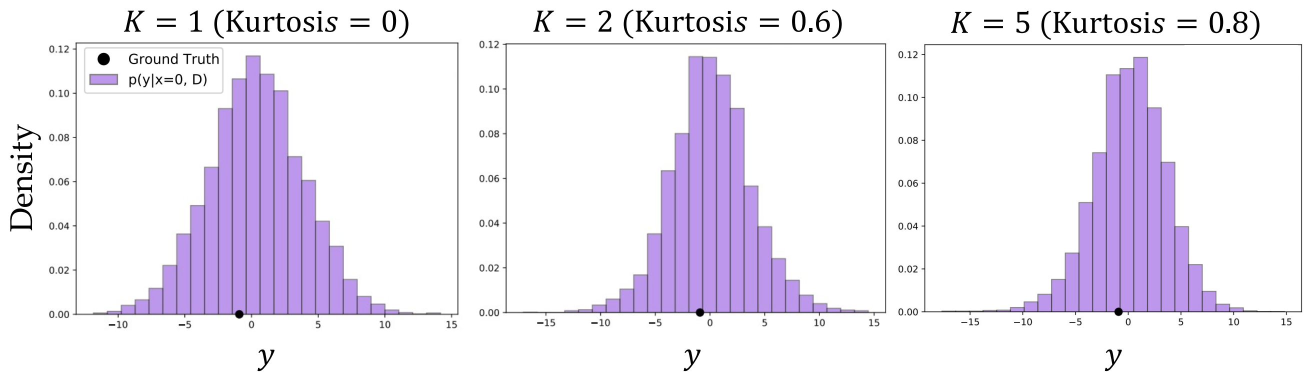

Figure 3: Effect of the number of mixtures on the learned kurtosis of the predictive distribution under heavy-tailed noise.

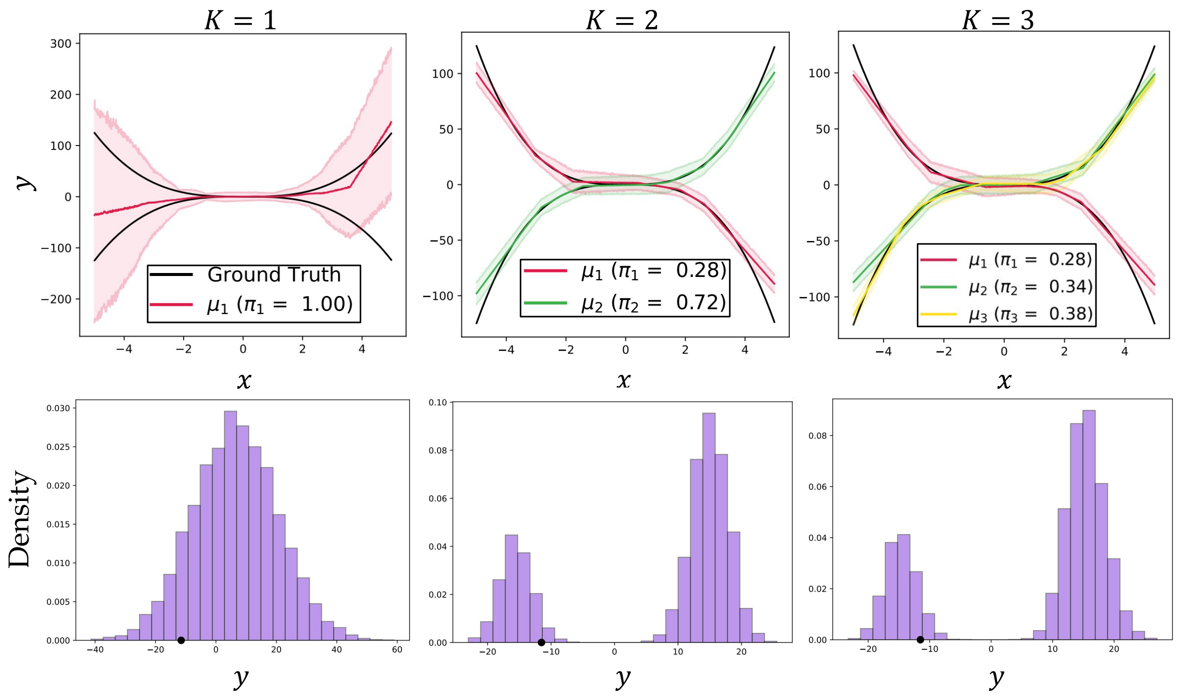

Figure 4: Effect of the number of mixture components on the learned predictive distribution under bimodal noise.

Figure 4 shows the effect of the number of mixture components on the learned predictive distribution in the case of the bimodal Gaussian. We can see that when DGMEs assumes only $K = 1$ mixture component, DGMEs have a similar predictive distribution as DEs, since the model will attempt to explain the bimodal data with a single Gaussian by overestimating the aleatoric noise. An interesting insight is that when DGMEs assume too many components (i.e., $K > 2$), the model is still able to accurately learn that the underlying predictive distribution is still bimodal.

### B.1.4 Ablation: Weight Initialization Schemes and Data Standardization

To test the impact of weight initialization of the neural network on the performance of DGME, we perform the following ablation study: we train a DGME for 5 rounds, where 10 epochs are used to resolve the M-Step in each round. We use the same architecture as in our toy experiments. We evaluate the NLL on the training set under five different initializations: PyTorch default initialization, initialization with uniform distribution with bounds -0.01 to 0.01, initialization with normal distribution with mean 0 and standard deviation $10^{-6}$, Xavier uniform initialization Glorot and Bengio [2010], and Xavier normal initialization. As a note, the PyTorch default initialization for a linear layer is done via a uniform distribution $\mathcal{U}(-\frac{1}{\sqrt{a}}, \frac{1}{\sqrt{a}})$, where $a$ denotes the number of input features to the linear layer. Please refer to Glorot and Bengio [2010] for more information on these weight initialization schemes.

| | PyTorch Default | Uniform | Normal | Xavier Uniform | Xavier Uniform |
|---|---|---|---|---|---|
| Normal - Unimodal | $2.86 \pm 0.07$ | $3.08 \pm 0.04$ | $3.02 \pm 0.04$ | $2.88 \pm 0.05$ | $2.90 \pm 0.05$ |
| Heavy-Tailed - Unimodal | $3.16 \pm 0.05$ | $3.42 \pm 0.06$ | $3.36 \pm 0.05$ | $3.18 \pm 0.05$ | $3.20 \pm 0.05$ |
| Normal - Bimodal | $3.36 \pm 0.17$ | $3.56 \pm 0.06$ | $3.61 \pm 0.06$ | $3.46 \pm 0.20$ | $3.47 \pm 0.20$ |

Table 3: Impact of different of weight initialization schemes on the train NLL when the data is not standardized.

| | PyTorch Default | Uniform | Normal | Xavier Uniform | Xavier Uniform |
|---|---|---|---|---|---|
| Normal - Unimodal | $2.55 \pm 0.02$ | $2.55 \pm 0.01$ | $2.56 \pm 0.01$ | $2.54 \pm 0.01$ | $2.53 \pm 0.01$ |
| Heavy-Tailed - Unimodal | $2.87 \pm 0.02$ | $2.88 \pm 0.01$ | $2.88 \pm 0.01$ | $2.86 \pm 0.01$ | $2.87 \pm 0.01$ |
| Normal - Bimodal | $3.13 \pm 0.07$ | $3.63 \pm 0.01$ | $3.60 \pm 0.04$ | $3.27 \pm 0.09$ | $3.24 \pm 0.09$ |

Table 4: Impact of different of weight initialization schemes on the train NLL when the data is standardized.

We train the model for each toy dataset over 20 total runs and report the average training NLL and its corresponding standard error. We run this ablation twice: once for training with non-standardized data and once for training with standardized data. The results are shown in Table 3 and Table 4. We can see from the results that although weight initialization has some impact on the results, if the data is standardized, it becomes less important. We also see that across all datasets, the default PyTorch initialization gives the most favorable results for both non-standardized and standardized data.

### B.1.5 Illustrative Results: Additive Gaussian Noise

We compare DGMEs with the baselines on the toy regression dataset with Gaussian noise. Figure 5 shows the performance of DGMEs compared to MDNs, MCD and DEs. DGMEs has comparable performance to MCD and DEs and outperforms MDNs.

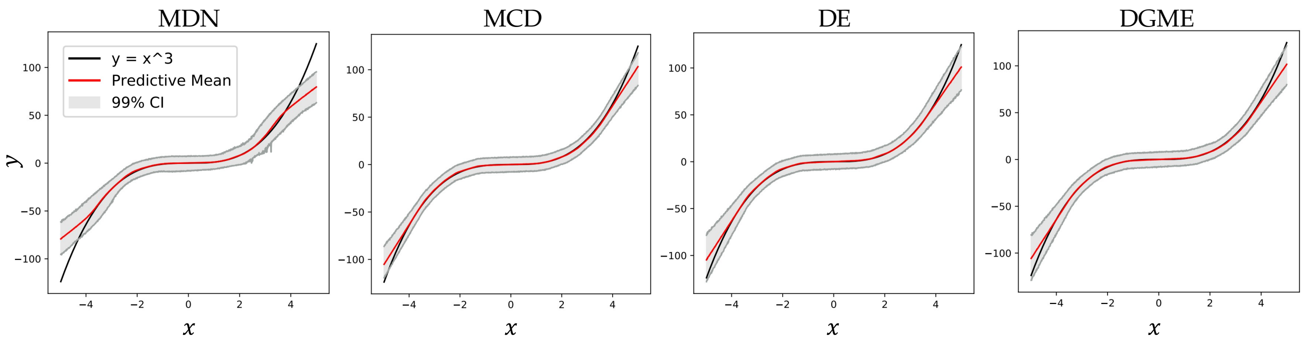

Figure 5: Performance on a toy regression task with Gaussian noise of DGMEs (right) with state-of-the-art methods MDNs, MCD and DEs.

## B.2 REGRESSION ON REAL DATASETS

For real data experiments on a regression task we use the following datasets: (a) Boston Housing dataset[1], (b) Concrete compressive strength dataset[2] [Yeh, 1998], (c) Energy efficiency dataset[3] [Tsanas and Xifara, 2012], (d) Kinematics of an 8 link robot arm dataset [4], (e) Combined cycle power plant dataset[5] [Tüfekci, 2014], (f) Wine dataset[6] and (g) Yacht

[1]https://www.kaggle.com/datasets/schirmerchad/bostonhoustingmlnd
[2]https://archive.ics.uci.edu/ml/datasets/concrete+compressive+strength
[3]https://archive.ics.uci.edu/ml/datasets/energy+efficiency
[4]https://www.openml.org/search?type=data&sort=runs&id=189&status=active
[5]https://archive.ics.uci.edu/ml/datasets/combined+cycle+power+plant
[6]https://archive.ics.uci.edu/ml/datasets/wine

Table 5: Test NLL for the regression experiments in the mixture of Gaussians case.

| Test NLL (Mixture of Gaussians) | | | | | | | |
|---|---|---|---|---|---|---|---|
| Dataset | MDNs | MCD | DEs | DGMEs (J=1) | DGMEs (J=2) | DGMEs (J=5) | DGMEs (J=10) |
| Boston housing | 2.71 ± 0.45 | 2.46 ± 0.25 | 2.41 ± 0.25 | **2.33 ± 0.18** | **2.33 ± 0.23** | 2.51 ± 0.33 | 2.74 ± 0.53 |
| Concrete | 3.04 ± 0.22 | 3.04 ± 0.09 | 3.06 ± 0.18 | 3.03 ± 0.10 | 2.99 ± 0.14 | 2.97 ± 0.24 | **2.94 ± 0.22** |
| Energy | **0.70 ± 0.17** | 1.99 ± 0.09 | 1.38 ± 0.22 | 1.56 ± 0.14 | 1.31 ± 0.12 | 0.96 ± 0.20 | 0.92 ± 0.48 |
| Kin8nm | -1.17 ± 0.04 | -0.95 ± 0.03 | -1.20 ± 0.02 | -1.20 ± 0.02 | -1.23 ± 0.03 | **-1.24 ± 0.02** | **-1.24 ± 0.02** |
| Power plant | **2.74 ± 0.04** | 2.80 ± 0.05 | 2.79 ± 0.04 | 2.81 ± 0.03 | 2.79 ± 0.03 | 2.77 ± 0.02 | 2.75 ± 0.02 |
| Wine | 0.43 ± 0.86 | 0.93 ± 0.06 | 0.94 ± 0.12 | 0.93 ± 0.12 | 0.90 ± 0.09 | 0.81 ± 0.11 | **0.18 ± 0.39** |
| Yacht | 0.51 ± 0.37 | 1.55 ± 0.12 | 1.18 ± 0.21 | 0.94 ± 0.19 | 0.66 ± 0.18 | 0.51 ± 0.23 | **0.42 ± 0.22** |

hydrodynamics dataset[7].

In the main text, to provide a fair comparison with techniques that assume the conditional distribution of the data is Gaussian, we summarize the mixture distribution output in both MDNs and DGMEs into a single Gaussian and then evaluate the NLL. This is analagous to the way DEs compute the NLL. We also provide additional results for the test NLL under the assumption of a mixture of Gaussians in Table 5 below.

### B.3 HYPERPARAMETER TUNING FOR FINANCIAL FORECASTING

We tuned the hyperparameters of the architecture (number of LSTM layers, number of fully-connected layers, number of LSTM hidden units, number of hidden units in fully-connected layers), optimization procedure (weight decay and learning rate), and the uncertainty quantification associated parameters (dropout probability, and homoscedastic variance value for MCD and MultiSWAG) for each of the approaches using cross validation . We note that all methods use the same feature extractor (LSTM and fully-connected network), which is obtained by hyperparameter tuning each dataset to a single network. To hyperparameter tune, we took the full training period and split it into an ordered sequence of a 90% training period and a 10% validation period. We select the hyperparameters based on the combination that maximizes the NLL on the validation period for each dataset.

## C POSSIBLE EXTENSION TO CLASSIFICATION TASKS

Techniques like MDNs and DGMEs are not suited for dealing with classification tasks, since the output of both models is a mixture of Gaussian distributions. For classification tasks, we instead can consider a mixture of categorical distributions, rather than a mixture of Gaussian distributions. In particular, the conditional distribution $p_\theta(y|x)$ is given by

$$p_\theta(y|x) = \sum_{k=1}^{K} \pi_k \prod_{i=1}^{d_y} p^i_{\theta_k}(x)^{\mathbb{I}(y=i)},$$

where $p^i_{\theta_k}(x)$ denotes the probability that $y$ belongs to the $i$-th class according to the $k$-th mixture. In this case, we assume MDNs and DGMEs output these probabilities rather than the mean and variance parameterizing a Gaussian distribution.

### C.1 ENTROPY CALCULATION

To evaluate uncertainty in classification tasks, we consider the average predictive entropy as the metric. To compute the average predictive entropy for a sample $x$, we use the following estimate:

$$\widehat{\text{Ent}}(x) = -\frac{1}{M} \sum_{m=1}^{M} \sum_{i \in \mathcal{C}} \tilde{p}^i_{(m)}(x) \log \tilde{p}^i_{(m)}(x),$$

[7]`https://archive.ics.uci.edu/ml/datasets/Yacht+Hydrodynamics`

| AVERAGE PREDICTIVE ENTROPY | | | | |
|---|---|---|---|---|
| Dataset | MDNs | MCD | DEs | DGMEs |
| MNIST (Known) | $0.019 \pm 0.005$ | $0.012 \pm 0.003$ | $0.012 \pm 0.002$ | $0.015 \pm 0.002$ |
| MNIST (Unknown) | $0.192 \pm 0.032$ | $0.180 \pm 0.020$ | $0.180 \pm 0.020$ | $0.193 \pm 0.016$ |
| Fashion-MNIST | $0.663 \pm 0.110$ | $0.714 \pm 0.140$ | $0.706 \pm 0.067$ | $0.698 \pm 0.057$ |

Table 6: Average predictive entropy for classification datasets. DGMEs are able to appropriately reason about the underlying uncertainty of OOD samples (MNIST with unknown classes and Fashion-MNIST) and is competitive with respect to state-of-the-art approaches.

where $\tilde{p}^i_{(m)}(x)$ denotes the probability of class $i$ according to the $m$-th sample from the predictive distribution and $\mathcal{C}$ denotes the set of classes. For both MDNs and DGMEs, these samples are obtained by the following procedure:

$$k^{(m)} \sim \text{Categorical}(\pi_1, \dots, \pi_K),$$
$$\tilde{p}^i_{(m)} = p^i_{\theta_{k^{(m)}}}.$$

Note that we incorporate dropout in the training procedure of MDNs and DGMEs for this experiment by applying a stochastic forward pass to the sampled network $k^{(m)}$.

### C.2 EXAMPLE: UNCERTAINTY EVALUATION ON MNIST

As an example, we compare DGMEs ability to reason about the underlying uncertainty of new samples with the baseline approaches with regards to the MNIST handwritten digits dataset. Specifically, for each method, we train a MLP network with 3 hidden layers and 200 hidden units per layer with ReLU activations on the MNIST dataset, including only digits 0-3 and 5-9. After the models are trained, we evaluate the average predictive entropy over three different datasets: the training dataset (known classes), a dataset containing only the digit 4 (unknown classes), and the Fashion-MNIST dataset (unrelated data). We use $M = 100$ samples from the predictive distribution to form an estimate of the predictive entropy for each method.We describe in more detail how the average predictive entropy is computed for each method in the Supplementary Material, Section B.3. The results for this experiment are shown in Table 6, which are averaged over 10 independent runs of each method. The results indicate that DGMEs are able to appropriately reason about the uncertainty in each of the datasets and is competitive with the baseline approaches in each case. DGMEs appropriately obtain that lowest entropy on the training dataset (i.e., the digits it was trained on), obtains a slightly higher entropy on the MNIST dataset containing unknown classes, and the highest entropy on the Fashion-MNIST dataset, which contains examples unrelated to the original classification task.

## D COMPARISON OF UNCERTAINTY QUANTIFICATION APPROACHES

Here, we provide an overall comparison of the benchmarks used in the experiments of this work as compared to the proposed approach along different qualities: the likelihood assumption, whether or not mixture weights are learned, how aleatoric uncertainty is quantified, and how epistemic uncertainty is quantified. This comparison is provided in Table 7

## E SAMPLING FROM THE PREDICTIVE DISTRIBUTION

To understand how sampling from the predictive distribution works in DGMEs, we begin with standard formula for determining the predictive distributions in Bayesian models:

$$p(y|x, \mathcal{D}) = \int_{\Theta} p_\theta(y|x) p(\theta|\mathcal{D}) d\theta.$$

In the case of DGMEs, $p_\theta(y|x)$ is a mixture of Gaussian distributions and $p(\theta|\mathcal{D})$ is approximated via dropout. An important property of the predictive distribution in the case of mixture distributions is that it can be expressed as a mixture of predictive

| **Method** | **Likelihood** | **Mixture Weights** | **Aleatoric Uncertainty** | **Epistemic Uncertainty** | **Other Notes** |
|---|---|---|---|---|---|
| MDNs | Mixture of Gaussians | Learned and input dependent | Heteroscedastic | None in original implementation, but dropout is applied for fair comparison in this implementation | Off-the-shelf can be applied to account for epistemic uncertainty (e.g., dropout, Laplace approximation, SWAG, variational Bayes, etc.) |
| MCD | Gaussian | Each prediction made via a stochastic forward pass at test time is equally weighted. | Homoscedastic | Dropout | |
| DEs | Gaussian | Assumed uniform | Heteroscedastic | Adversarial training and weight initialization. Dropout is also applied in this implementation using hyperparameter optimization. | |
| MultiSWAG | Gaussian | Assumed uniform | Homoscedastic | Stochastic weight averaging Gaussian (SWAG) | One can also account for heteroscedasticity by applying SWAG training to a deep ensemble that outputs a mean and variance |
| DGMEs | Mixture of Gaussians | Learned and independent of input | Heteroscedastic | Dropout in this implementation | Other methods to account for epistemic can be used off-the-shelf (e.g., Laplace approximation, SWAG, variational Bayes, etc.) |

Table 7: Summary of benchmarks as compared to DGMEs.

distributions. This property can be derived as follows:

$$\begin{aligned}
p(y|x,\mathcal{D}) &= \int_{\Theta} p_\theta(y|x)p(\theta|\mathcal{D})d\theta \\
&= \int_{\Theta} \left(\sum_{k=1}^{K} \pi_k p_k(y|x,\theta_k)\right) p(\theta|\mathcal{D})d\theta \\
&= \int_{\Theta} \sum_{k=1}^{K} \pi_k p_k(y|x,\theta_k)p(\theta|\mathcal{D})d\theta \\
&= \sum_{k=1}^{K} \pi_k \int_{\Theta} p_k(y|x,\theta_k)p(\theta|\mathcal{D})d\theta \\
&= \sum_{k=1}^{K} \pi_k \int_{\Theta_k} p_k(y|x,\theta_k)d\theta_k \underbrace{\int_{\Theta_{-k}} p(\theta|\mathcal{D})d\theta_{-k}}_{p(\theta_k|\mathcal{D})} \\
&= \sum_{k=1}^{K} \pi_k \int_{\Theta_k} p_k(y|x,\theta_k)p(\theta_k|\mathcal{D})d\theta_k.
\end{aligned}$$

Since $p_k(y|x,\mathcal{D}) = \int_{\Theta_k} p_k(y|x,\theta_k)p(\theta_k|\mathcal{D})d\theta_k$, we obtain the following expression for the predictive distribution:

$$p(y|x,\mathcal{D}) = \sum_{k=1}^{K} \pi_k p_k(y|x,\theta_k).$$

This form implies that we can draw samples from the predictive distribution in DGMEs using the following procedure:

1. Sample the mixture component:
$$k \sim \text{Categorical}(\pi_1, \dots, \pi_K)$$
2. Sample the posterior parameters of the given mixture component. In this work, dropout was used to approximate each posterior $p(\theta_k|\mathcal{D})$:
$$\begin{aligned} a_{k,i} &\sim \text{Bernoulli}(p_d), \quad i = 1, \dots, d_\theta, \\ \theta_k &= a_k \odot \theta_k^\star \end{aligned}$$
3. Draw the sample of $y$ from the appropriate predictive distribution:
$$y \sim p_k(y|x, \theta_k)$$

## References

Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332. PMLR, 2019.

Max H Farrell, Tengyuan Liang, and Sanjog Misra. Deep neural networks for estimation and inference. *Econometrica*, 89 (1):181–213, 2021.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

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

Athanasios Tsanas and Angeliki Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and buildings*, 49:560–567, 2012.

Pınar Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, 2014.

CF Jeff Wu. On the convergence properties of the em algorithm. *The Annals of statistics*, pages 95–103, 1983.

I-C Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998.