INTERPRETABLE MIXTURE DENSITY ESTIMATION BY USE OF DIFFERENTIABLE TREE-MODULE

Ryuichi Kanoh
Mobility Technologies Co., Ltd.
ryuichi.kano@mo-t.com

Tomu Yanabe
Mobility Technologies Co., Ltd.
tomu.yanabe@mo-t.com

ABSTRACT

In order to develop reliable services using machine learning, it is important to understand the uncertainty of the model outputs. Often the probability distribution that the prediction target follows has a complex shape, and a mixture distribution is assumed as a distribution that uncertainty follows. Since the output of mixture density estimation is complicated, its interpretability becomes important when considering its use in real services. In this paper, we propose a method for mixture density estimation that utilizes an interpretable tree structure. Further, a fast inference procedure based on time-invariant information cache achieves both high speed and interpretability.

1 INTRODUCTION

Machine learning (ML) is becoming more and more popular, and is being used in various services. When the outputs of ML are used to determine the behavior of a service, probabilistic prediction (Gal & Ghahramani, 2016; Lakshminarayanan et al., 2017; Duan et al., 2020) is attracting attention from the viewpoint of robustness. In the case of point prediction, the model's output is a single representative value. However, with the point prediction output, we cannot understand how confident the ML model is. Therefore, probabilistic prediction, in which the output is in the form of a probability distribution rather than a single value, has become an essential technique. Note that the probability distribution of targets does not always have a simple form (e.g., single Gaussian). In the real world, the target distribution is often a mixture distribution (e.g., Gaussian mixture). Although several ML models that use mixture distributions as output have been proposed, there are still some issues in terms of interpretability. In particular, since the output results are more complicated than point prediction or probabilistic prediction with a single distribution, understanding how mixture distributions are formed in the ML models becomes an important topic.

In this paper, we propose an interpretable method for the mixture density estimation using a tree-structured module. In particular, using the proposed model makes it easy to capture how a specific distribution is selected among multiple candidate distributions.

Assuming a real-world service, we consider a situation where the feature given as input to the ML model is a mixture of time-variant and time-invariant information[1]. Under such a situation, we propose a method that makes part of the inference process pre-computable by efficiently handling time-invariant input. It allows us to achieve both high interpretability and high speed.

2 RELATED WORK

2.1 MIXTURE DENSITY NETWORK

Mixture density network (MDN, Bishop (1994)) is an ML model that takes the output as formulated by a mixture of multiple distributions. Figure 1(a) shows an architecture of MDN. To represent a mixture distribution consisting of M Gaussians $\sum_{m=1}^{M} \alpha_m(\boldsymbol{x})\, \mathcal{N}\left(\mu_m(\boldsymbol{x}), \sigma_m(\boldsymbol{x})\right)$, MDN outputs

[1] Assume future vehicle counts on each road-segment is a target of the prediction. In this example, weather, temperature, and the number of surrounding vehicles can be time-variant information. In contrast, road length, number of lanes on the road, and road location are time-invariant information.

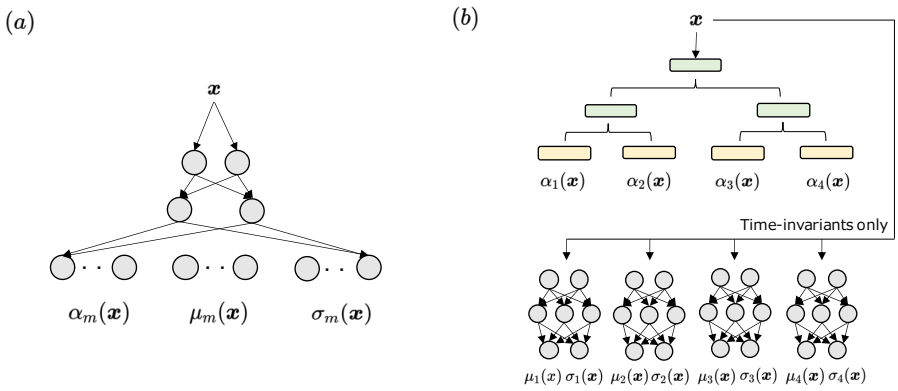

Figure 1: (a): Model architecture of Mixture Density Network. (b): Proposed model architecture.

a collection of M means $\mu_m(\boldsymbol{x})$, variances $\sigma_m(\boldsymbol{x})$ and weights $\alpha_m(\boldsymbol{x})$, where $\boldsymbol{x}$ is an input vector and $\mathcal{N}$ represents a Gaussian. The MDN has a structure like multi-layer perceptron (MLP), and the hidden layers are shared to the output of $\mu_m(\boldsymbol{x})$, $\sigma_m(\boldsymbol{x})$, and $\alpha_m(\boldsymbol{x})$. In practice, there are some constraints for parameters. First, weights must be normalized as $\sum_{i=1}^{m} \alpha_m(\boldsymbol{x}) = 1$. Therefore, the softmax function is applied for achieving constraints. Second, variances must not be negative. For this reason, the output of the model $\sigma_m(\boldsymbol{x})$ is exponentiated, and interpreted as variances. With obtained distribution parameters, negative log-likelihood (NLL) is minimized during the training.

Despite the simple structure, this MLP-based model does not the have high interpretability. Therefore, we propose a method that combines the benefits of the tree-structured model.

2.2 SOFT-TREE

A (hard) decision tree is an example of an ML model with high interpretability. However, since splitting search is a non-differentiable operation, it is not easy to combine differentiable models like a neural network. However, it is still possible to build a model that behaves like a decision tree using only differentiable operations (Frosst & Hinton, 2017).

Figure 2 shows a soft splitting operation. While hard decision trees decisively split data to either the right or the left, Soft-Trees splits data probabilistically. Each tree node i has a trainable filter $\boldsymbol{w_i}$ and a bias b_i. Splitting is expressed using

$$g_i = \varphi\left(\mathrm{softmax}\left(\boldsymbol{w}_i\right)\boldsymbol{x} + b_i\right), \qquad (1)$$

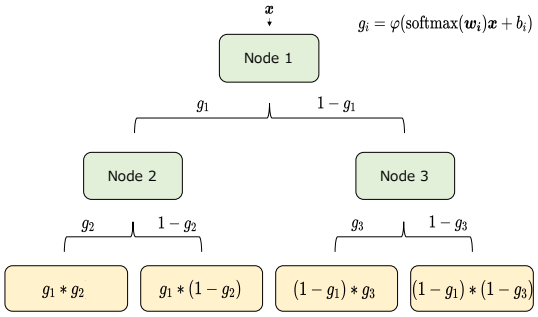

Figure 2: Soft splitting operation

where φ is an activation function like sigmoid that satisfies $0.0 \leq \varphi(x) \leq 1.0$, $\lim_{x\to\infty} \varphi(x) = 1.0$, and $\lim_{x\to-\infty} \varphi(x) = 0.0$. In analogy with the hard decision tree, $\mathrm{softmax}\left(\boldsymbol{w}_i\right)$ corresponds to the process of feature selection used for tree splitting[2], and b_i corresponds to the process of threshold determination for splitting. At each split, the data distribution split to the right and left node are $r_r = r_o * g_i$ and $r_l = r_o * (1 - g_i)$, where splitting node has r_o of data. If g_i is exactly 1.0 or 0.0, then the split process is equivalent to a hard decision tree. Performing this operation recursively, the leaves at the end of the tree-module have the data distribution corresponding to each leaf. This leaf distribution is used for weights on ensembling of the outputs of leaf-modules. The leaf-module can be a trainable constant scaler (that does not depend on $\boldsymbol{x}$) like a typical decision tree, or it can be another ML model like MLP.

[2]To emphasize the analogy of feature selection, filter weight $\boldsymbol{w_i}$ is often passed through an activation function such as softmax or entmax (Peters et al., 2019; Popov et al., 2020). Note that if $\mathrm{softmax}\left(\boldsymbol{w}_i\right)$ only contains 1.0 and 0.0, its role is equivalent to the hard feature selection.

Although such a tree-structured model has been proposed, how to perform mixture density estimation by using this model has not been studied well.

3 MIXTURE DENSITY ESTIMATION WITH TREE-MODULE

3.1 ARCHITECTURE

Figure 1(b) shows a proposed model architecture. Compared with a simple MDN (Section 2.1), there are two major differences. The first is the method for estimating $\alpha_m(\boldsymbol{x})$. While MDN outputs $\alpha_m(\boldsymbol{x})$ using the same procedure as $\mu_m(\boldsymbol{x})$ and $\sigma_m(\boldsymbol{x})$, the proposed method calculates $\alpha_m(\boldsymbol{x})$ using leaf data distribution of the Soft-Tree (Section 2.2). Because of this architecture, it is easy to capture how a particular distribution is selected among multiple candidate distributions. Second, while MDN shares the intermediate layers of a neural network and outputs multiple distribution parameters from it, in the proposed method, an MDN that outputs only one distribution is used for each leaf of the tree-module. This makes it easy to interpret what each leaf means. Since $M = 1$ for each leaf MDN, there is no need to consider $\alpha_m(\boldsymbol{x})$. Note that only time-invariant features are used for inputs of leaf-modules. The reasons are described in the following (Section 3.2).

3.2 FAST INFERENCE WITH PRE-COMPUTATION

Since many neural networks (MDNs that outputs parameters of a single distribution) are used for each leaf-module, the number of parameters of the whole model is larger than simple MDN. Therefore, even with interpretability, processing speed is expected to be slow. As a countermeasure, we use only time-invariant information as input of the leaf-modules. In that case, the leaf-module's output can be pre-computed, and only the tree-module needs to be calculated during the inference. Such a constraint achieves high-speed processing in real-world services where time-variant and time-invariant information are mixed.

Table 1 shows a comparison of the number of parameters in the model between Soft-Tree and MLP. Since the number of elements of the mixture distribution is not expected to be very large when predicting a mixture distribution, the tree's depth is expected not to be large. Assuming that the number of hidden units in MLP (W) is 100, the number of input features is 30, the number of output values is 2, and the depth of MLP (L) and Soft-Tree (D) is 3, then the number of parameters in Soft-Tree is 210, and the number of parameters in MLP is $13,200$. From this example, we see that the amount of parameters in the tree-module required for actual inference is not large. There are techniques to reduce the amount of calculation for Soft-Tree, that avoids exponential order computational complexity. In addition, we can speed up the training process by making splitting sparse like a hard decision tree. Details are in Appendix A.

model	# parameters
Soft-Tree	$\text{len(input)}(2^{D+1} - 1)$
MLP (e.g., leaf-modules)	$W(\text{len(input)} + \text{len(output)}) + W^2(L - 2)$

Table 1: A number of parameters for L layer MLP with W hidden units per each layer, and Soft-Tree with D depth. The Soft-Tree is assumed to be a complete binary tree. For simplicity, the number of bias terms is not counted. We assume W does not change for each layer.

4 NUMERICAL EXPERIMENTS

4.1 SETUP

We use real-world data obtained by MOV[3] (renamed as GO[4] in September 2020), a Japanese taxi ride-hailing service by Mobility Technologies[5]. In this experiment, the time duration between entering and exiting the road-segments of taxi stands is the prediction target. We made a prediction

[3]https://m-o-v.jp/

[4]https://go.mo-t.com/

[5]https://mo-t.com/

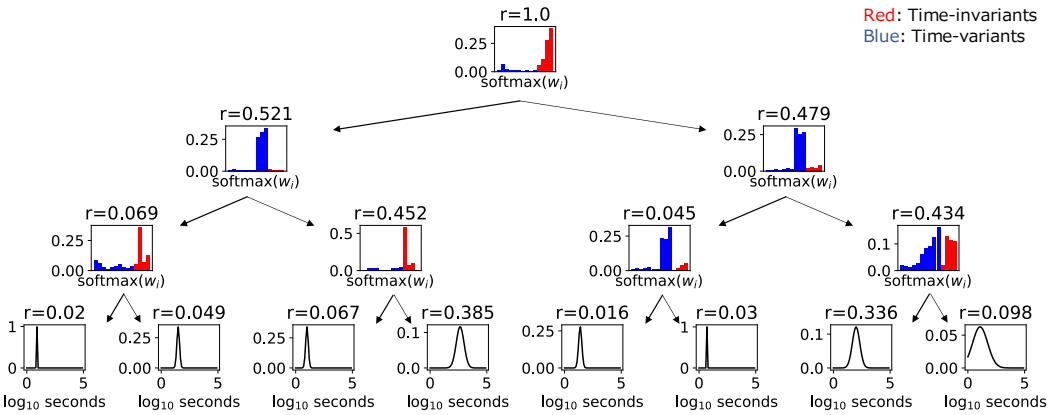

Figure 3: Trained proposed model. For a tree-module, the length of each bar is the softmax($\boldsymbol{w_i}$). The red and blue colors in the bar plots correspond to softmax($\boldsymbol{w_i}$) for time-invariant and time-variant features, respectively. For leaf-modules, their output probability distribution for a single input feature set is plotted. At the top of each plot, how much percentage is flowing to that node is stated. The softmax($\boldsymbol{w_i}$) does not change when the input data changes. In contrast, the leaves' information, such as the probability distribution and how much data is distributed to each node, changes with input data.

for each record that occurs each time a vehicle passes by. Since the target does not take negative values and is represented by a superposition of lognormal distributions, the logarithm is taken to the prediction target. Other details of experiments are in Appendix B.

4.2 Result

	model	NLL (lower is better)
(1)	MDN	-0.1937 ± 0.2144
(2)	Soft-Tree with constant leaves	0.9926 ± 0.1148
(3)	Proposed	-0.4512 ± 0.1925

Table 2: Comparison of probabilistic regression performance as measured by NLL. The mean and standard deviation of the NLL are summarized from experiments with 10 different random seeds.

Table 2 shows a performance comparison. As benchmarks, we used (1) MDN shown in Figure 1(a), and (2) Soft-Tree with constant leaves that do not depend on $\boldsymbol{x}$. As for (2), the difference from the proposed method is that the output of the leaf-modules (μ_m and σ_m) do not depend on any input $\boldsymbol{x}$ during the inference, and therefore the candidates of the element of mixture distributions are common for all prediction results. Note that in the proposed method, the output of the leaf-modules depends on the time-invariant input features. On the one side, compared to (1), the performance of the proposed method is better, even though the number of the parameters required for inference is smaller. On the other hand, compared to (2), we can see the importance of changing the candidate elements of the mixture distribution for each input. Note that the computational complexity of (2) and the proposed method remains the same in inference due to the pre-computation.

Figure 3 shows a trained proposed model. As shown in the figure, it is possible to visually check which features are responded to and how the data branched out and fell on the leaves. Each node of the tree-module has different features to focus on, indicating that each node has a different role. Since the leaf-modules that outputs a single Gaussian take only time-invariant features as input, the Gaussian visualized in the leaf-module does not change even if the time-variant features change. It is beneficial from the point of view of speed (Section 3.2) and interpretability. Since the role of each leaf does not change with time-variant input, it makes it easy to interpret when making comparisons between prediction outputs with the same time-invariant features (e.g., prediction outputs for the same road-segment at different times).

REFERENCES

Christopher M. Bishop. Mixture density networks. Technical report, 1994.

Tony Duan, Avati Anand, Daisy Yi Ding, Khanh K. Thai, Sanjay Basu, Andrew Ng, and Alejandro Schuler. NGBoost: Natural gradient boosting for probabilistic prediction. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020.

Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. *CoRR*, 2017.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The Tree Ensemble Layer: Differentiability meets Conditional Computation. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, 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 *International Conference on Learning Representations*, 2021.

Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. In *In Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2009.

Ben Peters, Vlad Niculae, and André F. T. Martins. Sparse sequence-to-sequence models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *International Conference on Learning Representations*, 2020.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, 2018.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations*, 2017.

Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive Neural Trees. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

A ADDITIONAL SPEEDUP FOR TREE-MODULE

Sparse Split A typical decision tree is a binary tree, where 100% of the data is assigned to either the right node or the left node at the time of splitting. Consider we have a complete binary hard decision tree that has a depth of 4. Although it has 15 nodes in total, the split operation is not required for all nodes because of this sparszity. Soft-Tree splitting does not always flow completely to either node, but there are a number of studies for achieving sparse splitting. For example, modifying the activation function φ used in the splitting (Equation (1) for taking the value of 0.0 or 1.0 strictly is one strategy. Notably, sparsity can achieve not only faster inference, but also faster training. (Shazeer et al., 2017; Hazimeh et al., 2020; Lepikhin et al., 2021). If the data distribution assigned to a leaf is exactly zero, the leaf-module calculation becomes unnecessary, and a large computational savings can be achieved.

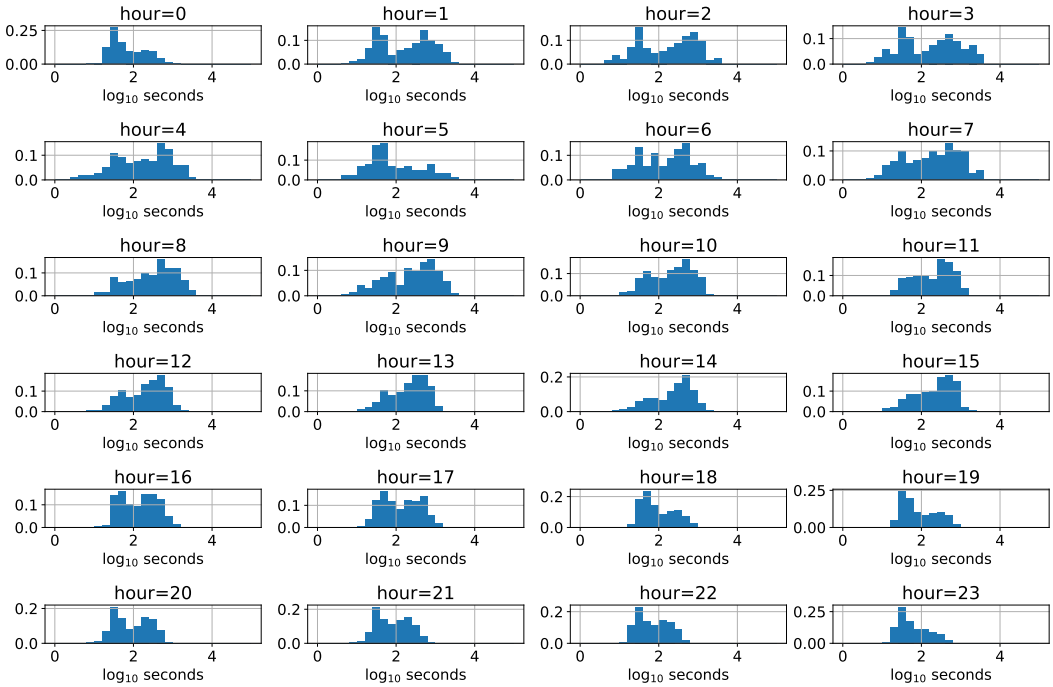

Figure 4: Distribution of the prediction target by an hour in a day in a road-segment

Oblivious Tree An oblivious decision tree (Popov et al., 2020; Prokhorenkova et al., 2018) uses the same splitting criterion across an entire level of the tree. Because of its structure, the number of split process in the Soft-Tree can be reduced to the decision process for the depth of the tree.

Architecture Search In a general decision tree, the structure is not determined before training, and the tree grows during the training. As an analogy, Tanno et al. (2019) proposed a method for growing Soft-Tree during training. There is a possibility that it allows us to use a smaller tree structure than the one we had pre-determined.

B Details of numerical experiments

Dataset Vehicle position (GPS latitude, longitude), and their status (Looking for passengers, carrying passengers, heading to reservation location) are obtained every 3 seconds for each taxi. Hidden Markov map-matching (Newson & Krumm, 2009) is used for all GPS points. For predicting the time duration between entering and exiting the road-segments of taxi stands, we use taxi stands that exist within a square with a side length of $10\mathrm{km}$, centered at latitude $= 35.4583$ and longitude $= 139.5625$, which vehicles passed through more than $10,000$ times in a month. In total, $939,954$ records are used for numerical experiments. Figure 4 shows an example of the distribution of the prediction target by an hour in a day in a road-segment.

Training procedure Numerical experiments are conducted with the data acquired in January 2020. Datasets obtained on January 1-14, 14-21, and 21-28 are used as train, valid, and test dataset, respectively. The performance is evaluated using the test data, and the values are reported when the training had the best performance for the valid data. We perform 2.0×10^3 optimization steps for minimizing the NLL. The prediction target is log-scaled. Batch size and learning rate are 2048 and 1.0×10^{-1}, respectively. We use Adam optimizer (Kingma & Ba, 2015) with a cosine-annealing learning rate scheduler.

Model Architecture We use a Soft-Tree with a depth $D = 3$ and an MLP with depth $L = 2$ and width $W = 50$ as the leaf-modules. Since the depth of the Soft-Tree is 3, it can represent a mixture distribution consisting of at most $2^3 = 8$. In order to align the conditions, the benchmark MDN (Table 2) also outputs parameters of 8 Gaussians. W and L are the same as benchmark MDN and proposed method.

Input features 14 features are used for training and inference. There are 10 time-variant features and 4 time-invariant features.

- Time-variant features
  1. Sine/Cosine transformed hour of a day
  2. Sine/Cosine transformed day of week
  3. Passed vehicle counts in recent 15min/30min/60min
  4. Average time duration for passing the taxi stands in recent 15min/30min/60min
- Time-invariant features
  1. Latitude and longitude of the taxi stand
  2. Length of the road-segment of the taxi stand
  3. Total vehicle counts per taxi stand

All features are numerical. Standard scaling is applied for each feature to achieve zero-mean and unit-variance.