DOI:10.1190/image2022-3729992.1

Using Mixture Density Networks for Uncertainty and Prediction in Seismic Reservoir Characterization

Cornelius Rosenbaum[*], *Ryan Warnick*[*], *Anar Yusifov*[†], *Reetam Biswas*[†], *and Atish Roy*[†]
[*] *Numerical Algorithms Group*
[†] *bp*

SUMMARY

Uncertainty quantification has received significant focus in the geophysics community in recent years; extending from general simulation based approaches to measure the amount of variation in the system under study, as well as more complex Bayesian modalities which incorporate ideas from Bayesian statistics and many other methods not mentioned here. More recently, there has been work in Bayesian approaches to neural networks, which endow the parameters of neural networks with prior distributions and exploiting the flexibility of neural network architectures to model complex behavior. We present an illustration of a more recently developed style of modeling uncertainty, Mixture Density Networks (MDNs), and present an application of the method to a reservoir characterization problem. Our approach provides good performance in prediction of the reservoir properties of three of the relevant reservoir rock properties with corresponding uncertainties. In addition, the approach permits new types of inferences that can help guide practitioners in their study of the reservoir.

INTRODUCTION

Since being introduced and refined in the 80's, neural networks (NNs) have emerged as a dominant force in the machine learning and data science community for modeling complex relationships between data. Recently, many APIs have been developed to expedite implementation and exploit computational resources in various programming languages, and this has accelerated acceptance of NNs in a wider audience of users. Variations on the standard NN template have been developed to accomplish tasks in a number of a different problem settings; including classification (Miller et al., 1995), Natural Language Processing (Morchid, 2019), computer vision (Khan et al., 2018), and many others. The ubiquity of NNs in many modelers toolboxes has led them to become in many ways a de facto standard for comparison with competing methods.

In the last few decades, ideas towards incorporating uncertainty into NNs have been proposed. These include Bayesian NN's (Mullachery et al., 2018), where the parameters of the network itself are given priors, MDNs, which we discuss further below and illustrate in this abstract, Variational Autoencoders (Doersch, 2016), which use variational methods (Fox and Roberts, 2012) to construct a generative model using a NN, and others. Using the generality of NNs for approximating large classes of nonlinear functions, and recent work on computational tools and methods in probabilistic modeling, has opened the door to many inventive approaches towards modeling complex phenomena.

MDNs were developed by Bishop (1994) as one of the earlier approaches to incorporating uncertainty into neural networks, but did not become prominent until more recently, when software for their implementation became available. With the data science revolution going on now across a vast swath of disciplines, interest in these methods has grown and applications can be found across a variety of domains. The general idea behind MDNs is to use a NN, but instead of modeling the output/dependent variables, henceforth called output variables, as the output layer of the NN, we estimate parameters of a distribution from which the output variables are generated from. Commonly this output distribution is modeled as a mixture distribution, though this is not necessary, to permit a higher level of flexibility in the distribution over the output variables. This mixture model approach is particularly helpful in geophysical applications, where the inversion framework is often ill-posed, as there could be many possible solutions for the variables the network is estimating which map to the correct input. Thus using a mixture over possible solutions to the problem exposes multiple available solutions.

We give an introduction to the theory of MDNs, with some assumptions about a background in neural networks (see Hassoun et al. (1995) for a thorough review of the basics), followed by our application of an MDN to an inverse problem in reservoir characterization. We applied the MDN technique to the geophysical problem since most of the conventional geophysical inversion workflows are deterministic and do not consider non-uniqueness in predictions. Inversion from seismic amplitude is an ill-posed problem and so we should not only find ways to move away from a single prediction, but also use a robust algorithm to build confidence in the uncertainty. We implement the model in tensorflow-probability (Dillon et al., 2017), a Probabilistic Programming Language (Goodman, 2013) which is integrated with tensorflow (Abadi, 2016) and keras (Ketkar, 2017), allowing us to implement the MDN with minimal effort and use GPUs to expedite training. In the theory section we go over the basic changes to the loss function required for a MDN, and the flexibility of the approach.

THEORY

Background

Neural Networks

The first important concept for MDNs is neural networks, and we give a brief introduction here, omitting many of the details.

Suppose we have input data $\{X_i\}_{i=1}^n \subset \mathscr{X}$ and corresponding output data $\{Y_i\}_{i=1}^n \subset \mathscr{Y}$, where each X_i is mapped to a corre-

sponding Y_i through some (not necessarily linear) relationship, $f : \mathscr{X} \to \mathscr{Y}$. $\mathscr{X}$ and $\mathscr{Y}$ could be spaces with various forms, including $\mathbf{R}$ for scalar values, $\mathbf{R}^K$ for some $K \in \mathbf{N}$ for vectors, or, in the case of 2D images, $\mathbf{R}^{K\times L}$ for $K, L \in \mathbf{N}$.

Neural networks attempt to solve the problem of estimating a suitable f from the data, such that for unobserved values $X^* \in \mathscr{X}$ we can predict a corresponding output value with minimal error, $Y^* \in \mathscr{Y}$. This is done by constructing a network of compositions of simple functions together into a composite function $\hat{f}(\circ;\eta) : \mathscr{X} \to \mathscr{Y}$, with the set of all individual simple functions being parameterized by a parameter vector η. To estimate η, and thus estimate $\hat{f}$, we minimize some loss function $L(\hat{Y}, Y)$; where $\hat{Y} \in \mathscr{Y} = \hat{f}(X;\eta)$ and minimization with respect to η.

Probability Density Function

Another important concept for MDNs is the idea of a probability density. A parameterized probability density $P(y|\theta)$, where θ is a fixed parameter, is a function defined on a set $\mathscr{Y}$, such that $P(y|\theta)$ is positive for all $y \in \mathscr{Y}$ and $\int_{y\in\mathscr{Y}} P(y|\theta)dy = 1$. For any subset $\mathbf{Y} \subset \mathscr{Y}$, the probability of the event $\mathbf{Y} \subseteq \mathscr{Y}$ is equal to $P(\mathbf{Y}|\theta) = \int_{y\in\mathbf{Y}} P(y|\theta)dy$. We denote by $y \sim P(y|\theta)$ that y follows the distribution implied by the density $P(y|\theta)$. One ubiquitous example of a probability density is the density of a Gaussian distribution:

$$\mathrm{N}(y|\mu,\sigma^2)) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp[-\frac{1}{2}(\frac{(y-\mu)^2}{\sigma^2})] \tag{1}$$

where $\theta = (\mu, \sigma^2) \in \mathbf{R} \times \mathbf{R}^+$.

If one has observed data, the "likelihood" is the function obtained when one holds the observation y fixed, and treats the density as a function of θ. That is, $P(y|\theta) = L(\theta;y) : \theta \to \mathbf{R}^+$. Maximizing the likelihood as a function of θ (conversely, minimizing the negative likelihood) is a common method for inferring the value of the parameter θ. Due to the monotonicity of the log function, minimizing the negative log of the likelihood gives the same result. Note that if you have multiple data points $\{y_i\}_{i=1}^n$ which are independent of each other (a strong assumption), then the distribution is the product of the density functions, $P(\{y_i\}_{i=1}^n|\theta) = \prod_{i=1}^n P(y_i|\theta)$, and then the negative log likelihood is the sum of the individual negative log likelihoods: $NLL(\theta|\{y_i\}_{i=1}^n) = \sum_{i=1}^n -\log P(y_i|\theta)$. This convenient trick is important for implementing the MDN we describe below.

Mixture Distributions

The final relevant point is a mixture distribution. Suppose we have a probability vector $\alpha \in [0,1]^K$ such that $\sum_{k=1}^K \alpha_k = 1$, and a set of parameters for some parametric family of distributions: $\{\theta_k\}_{k=1}^K$. Then one can construct a mixture distribution with the associated mixing probabilities α and parameters $\{\theta_k\}_{k=1}^K$, represented as the sum of each parameterized distribution weighted by it's associated probability vector component:

$$P(y|\alpha,\{\theta_k\}_{k=1}^K) = \sum_{k=1}^K \alpha_k P(y|\theta_k) \tag{2}$$

To get some intuition about this, suppose we want to sample from $P(y|\alpha,\{\theta_k\}_{k=1}^K)$; we would first sample $j \sim \mathrm{Categorical}(\alpha)$ and then sample $y \sim P(y|\theta_j)$. We see then that each element of $\{P(y|\theta_k)\}_{k=1}^K$ is weighted by the corresponding probability in the mixture, defining a distribution that is in some sense the weighted sum of individual distributions.

Mixture Density Networks

MDNs meld these three ideas together, such that instead of modeling the observation data $\{Y_i\}_{i=1}^n$ as the output of a deterministic function, $Y = f(X)$, we model $Y \sim P(Y|\theta(X))$; where $P(Y|\theta)$ is some parametric family of distributions, with the estimated parameter θ modeled as a function of X. The function through which the parameters are estimated, $\theta(X)$, is modeled using a neural network, $\hat{\theta}(X;\eta)$ as described in the Background section, and we seek to estimate the parameters η to represent the function $\theta(X)$ as closely as possible with $\hat{\theta}(X;\eta)$. It is common to model the parameterized family as a family of mixtures; such that $\theta = (\alpha, \{\theta_k\}_{k=1}^K)$.

Modeling the network in this way, we can see it is a two step process: for a given estimated function $\hat{\theta}(X;\eta)$ we pass X into $\hat{\theta}(X;\eta)$ to obtain an estimate for the parameters $\hat{\theta}$, and then $y \sim P(y|\hat{\theta})$. This creates a different distribution on each Y created by the corresponding X. This means that a MDN estimates a different conditional distribution for each Y conditional on it's corresponding X; $P(Y|X)$, where the relationship is modeled using a parametrized family of distributions and the parameters are estimated as a function of X; with the structure of this function estimated using a NN. Figure 1 gives an example of a mixture density network for univariate normal distributions.

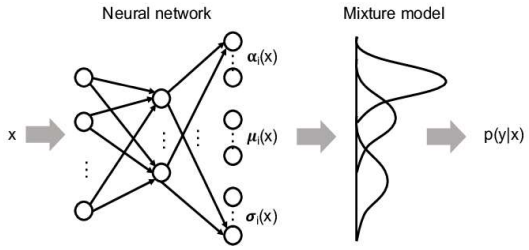

Figure 1: Diagram of a mixture density network for univariate Gaussian distributions. Note that we estimate multiple μ and σ values for each mixture component, and the probability vector $\{\alpha_k\}_{k=1}^K$.

Our Model

Distribution of the Output Data

Our output data, $\{Y_i\}_{i=1}^n \in \mathbf{R}^P$, are P length vectors representing some reservoir attribute at a given location; for example, porosity, shale volume, or impedance. Our family of distributions for these output values is a multivariate Gaussian dis-

tribution (MVN), with probability density defined for $Y \in \mathbf{R}^P$ specified as:

$$\mathrm{MVN}(Y|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^p \det(\Sigma)}} \exp[-\frac{1}{2}(y-\mu)^T \Sigma^{-1}(y-\mu)] \tag{3}$$

Where $\mu \in \mathbf{R}^P$, and $\Sigma \in \mathbf{R}^{P\times P}$ is a symmetric positive definite covariance matrix. We generalize this by assuming that Y comes from a mixture of MVNs, specifically, for $\theta = \{\alpha, \{(\mu_k, \Sigma_k)\}_{k=1}^K\}$, where $\alpha \in [0,1]^K$ and $\sum_{k=1}^K \alpha_k = 1$, we have:

$$P(Y|\theta) = P(Y|\alpha, \{(\mu_k,\Sigma_k)\}_{k=1}^K = \sum_{k=1}^K \alpha_k \mathrm{MVN}(Y|\mu_k,\Sigma_k) \tag{4}$$

We then want to estimate $\hat{\theta}(X;\eta)$, where $\hat{\theta} \approx \theta$. Mixing together MVNs like this allows us to better model complex distributions on $\mathbf{R}^P$ than using a single MVN individually.

Neural Network

Our input data space is $X \in \mathbf{R}^{Q\times M}$, where X is an array of Q seismic synthetics of length M. The first step in our network is to pass each $\{X_{q\circ}\}_{m=1}^M$ through a 1D convolution layer. The resultant convolutions are then stacked together and passed through a 2D convolution, followed by a flattening layer.

The layers beyond the flattening layer are where the network gets interesting. We need to estimate $\{\alpha, \{\mu_k \Sigma_k\}_{k=1}^K\}$. To estimate α we use a Softmax layer, the canonical layer for estimating probability vectors. To estimate $\{\mu_k\}_{k=1}^K$, we estimate each individual $\mu_k \in \mathbf{R}^P$ as the output of a dense layer with P values and a linear activation, and repeat this K times.

The estimation of the symmetric positive definite matrices $\{\Sigma_k\}_{k=1}^K$ is trickier. Note that the covariance matrix is symmetric, and that it is fully defined by the lower triangular elements, denote these Σ_L. Then note that a covariance matrix can be constructed out of a correlation matrix, $\rho_L \in \mathbf{R}^{P\times P}$, where $\rho_{L_{ij}}$ corresponds to the correlation between values i and j and is zero on the upper triangular elements and one on the diagonal, and a matrix of zeros with the standard deviations along the diagonal, $S \in \mathbf{R}^{P\times P}$. Then $\Sigma_L = S\rho_L S$. This ensures that the covariance matrix is positive semi-definite. This is illustrated in Equation 6 for a $P \times P$ covariance matrix.

$$\Sigma_L = \begin{bmatrix} \sigma_{11}^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ \sigma_{PP}\sigma_{11}\rho_{P1} & \dots & \sigma_{PP}^2 \end{bmatrix} = S\rho_L S \tag{5}$$

$$= \begin{bmatrix} \sigma_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_{PP} \end{bmatrix} \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ \rho P1 & \dots & 1 \end{bmatrix} \begin{bmatrix} \sigma_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_{PP} \end{bmatrix} \tag{6}$$

We can construct ρ_L be using a dense layer with $\frac{P*(P-1)}{2}$ elements and a Tanh activation function. The Tanh activation function is bounded between [-1,1], as are correlations. The diagonal elements of S are constructed out of what we term a non-negative ELU activation, NNELU, which is equal to a ELU activation plus one. We estimate P of them. Once ρ_L and S are constructed, we use the formula 6 to estimate Σ_L, which fully defines Σ. We repeat this process K times to construct all of the individual $\Sigma_k, k \in \{1,...,K\}$.

Loss Function

Our loss function is the negative log likelihood of the observation Y_i given X_i. That is, using $\hat{\theta}(X;\eta)$ to estimate our parameters $\hat{\alpha}, \{\hat{\mu}_k, \hat{\Sigma}_k\}_{k=1}^K$, we plug these parameters into $P(Y_i|X_i,\hat{\theta})$, and minimize over η, by optimizing:

$$\mathrm{argmin}_\eta NLL(\hat{\theta}(\{X_i\}_{i=1}^n;\eta);\{Y_i\}_{i=1}^n) \tag{7}$$

Using the likelihood of our mixture of MVNs to construct the loss function, our model is completed. By doing things in this way, we incorporate the probability model into the specification of the mixture distribution and the loss function, while still retaining the training procedures of a standard neural network. Training, testing, and validation sets are used, as is common in neural network training, by constructing a 1-1 partition of the sets $\{X_i\}_{i=1}^n$ and $\{Y_i\}_{i=1}^n$.

Implementation

We implement the model in tensorflow-probability (tfp), using tensorflow and keras to perform the optimization and neural network building, respectively. The mixture distribution is constructed out of a tfp.distributions distribution object; which creates a mixture of the same family of MVNs.

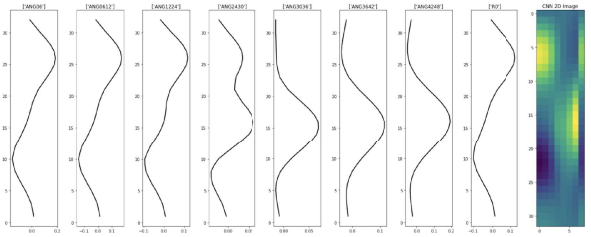

Figure 2: Plot showing an example of the synthetics passed into the neural network. The far right plot is the synthetics stacked together into a heatmap.

Our input data $\{X_i\}_{i=1}^n \subset \mathbf{R}^{7\times 32}$ is a set of 7 synthetics of length 32 for each observed X_i, and our output data $\{Y_i\}_{i=1}^n \subset \mathbf{R}^{3\times 18}$. Figure 2 illustrates the synthetics passed into the model for a particular synthetic set. Each output data is composed of three seismic attribute traces of length 18; specifically, Shale Volume, Porosity, and Background Impedance. We estimate traces of length 18 samples for each of the three reservoir properties separately, with a cumulative loss function constructed as the same of the individual losses. The full network architecture, following what was outlined in the neural networks subsection of the Our Model section, is illustrated in Figure 3.

We estimate $K = 10$ mixture components; creating 10 means and 10 covariance matrices; as well as a probability vector of length 10.

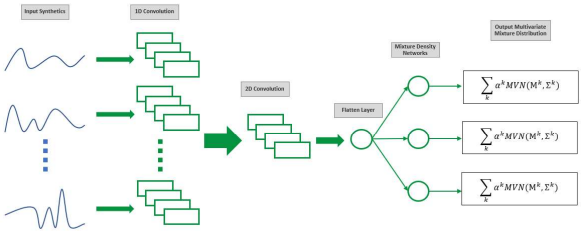

Figure 3: Diagram of the neural network architecture for estimating the distribution over our three seismic attributes.

As a Proof of Concept, the initial dataset used in this analysis is a synthetic. We create 59686 pseudo-wells from one real field as an analog. The suite of logs, including Vp, Vs, and density, are created for each of the pseudo-wells. Then each of the pseudo-wells are forward modeled with the average wavelets from seismic to create the synthetic seismogram for seven different angle stacks and the intercept data.

RESULTS

With these multi-attribute synthetics we train with their corresponding pseudo-well logs, Shale Volume, Total Porosity, and Impedance, as the labels along the respective distributions from the real analog well. The predictions based on the MDN are not one rock property estimate for a given synthetic, but a distribution of prediction based on the input observed distribution of the real analog well.

Figure 4 shows the predicted distribution over each of the three predicted rock properties. Note that these uncertainties are distributed according to a MVN, so there are correlations in the uncertainties across the traces. Figure 5 illustrates the covariance relationships in each component of the mixture. Note that three of the mixture component covariances have a specific block structure, while other components have a tendency to capture more distinct covariances along the attribute trace.

Figure 5 also indicates how much probability is given to each mixture component; that is the values of $\{\alpha_k\}_{k=1}^{K}$ in the mixture outlined in Equation 4.

DISCUSSION AND CONCLUSION

We utilize a probabilistic programming language to implement a MDN in the context of a geophysical inverse problem in reservoir characterization. We haveshown this tool as an effective mechanism for inference using probabilistic concepts incorporated into a neural network, all while providing new types of inferences through the estimation of the mixture component probabilities and covariances. Additionally, using readily available computational tools makes this type of problem easily approachable to people less familiar with the specifics of how to implement probabilistic models.

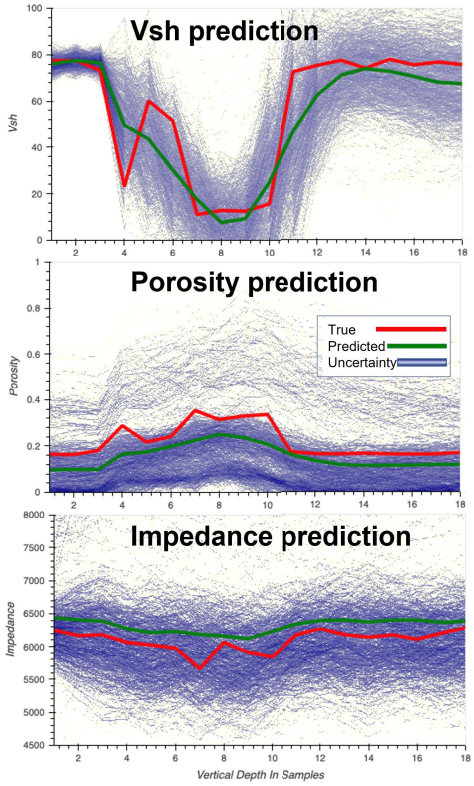

Figure 4: Plots showing predicted trace values with uncertainty. Uncertainty includes solutions to the ill posed problem which are estimated from the different mixtures in the MDN. The true trace value is given in red, and the predicted mean of the mixture is given in green. The colors are adjusted to correlate with the uncertainty around a particular region; with light yellow corresponding to less likely and dark blue for highly likely.

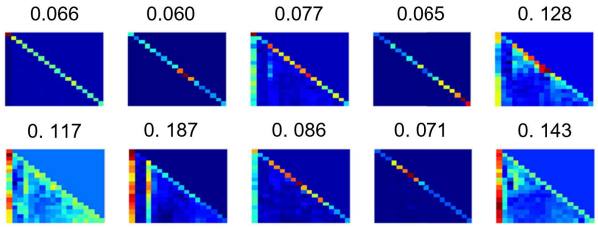

Figure 5: Estimated covariance matrices of each mixture component out of 10 mixture components, with associated mixture probabilities; done for an individual synthetic set for porosity.

REFERENCES

Abadi, M., 2016, TensorFlow: Learning functions at scale: Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming.

Bishop, C. M., 1994, Mixture density networks.

Dillon, J. V., I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous, 2017, Tensorflow distributions: arXiv preprint, arXiv:1711.10604.

Doersch, C., 2016, Tutorial on variational autoencoders: arXiv preprint, arXiv:1606.05908.

Fox, C. W., and S. J. Roberts, 2012, A tutorial on variational Bayesian inference: Artificial Intelligence Review, **38**, 85–95, doi: https://doi.org/10.1007/s10462-011-9236-8.

Goodman, N. D., 2013, The principles and practice of probabilistic programming: ACM SIGPLAN Notices, **48**, 399–402, doi: https://doi.org/10.1145/2480359.2429117.

Hassoun, M. H., 1995, Fundamentals of artificial neural networks: MIT Press.

Ketkar, N., 2017, Introduction to keras, *in* Deep learning with Python: Springer, 97–111.

Khan, S., H. Rahmani, S. A. A. Shah, and M. Bennamoun, 2018, A guide to convolutional neural networks for computer vision: Synthesis Lectures on Computer Vision, **8**, 1–207, doi: https://doi.org/10.2200/S00822ED1V01Y201712COV015.

Miller, D. M., E. J. Kaminsky, and S. Rana, 1995, Neural network classification of remote-sensing data: Computers & Geosciences, **21**, 377–386, doi: https://doi.org/10.1016/0098-3004(94)00082-6.

Morchid, M., 2019, Neural networks for natural language processing: Ph.D. thesis, Avignon Universite.

Mullachery, V., A. Khera, and A. Husain, 2018, Bayesian neural networks: arXiv preprint, arXiv:1801.07710.