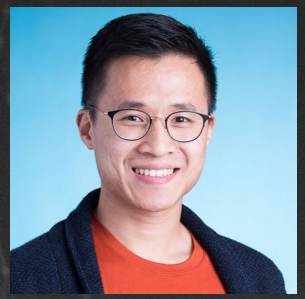
Hybrid Models with Deep and Invertible Features

Eric Nalisnick*, Akihiro Matsukawa*,
Yee Whye Teh, Dilan Gorur, Balaji Lakshminarayanan

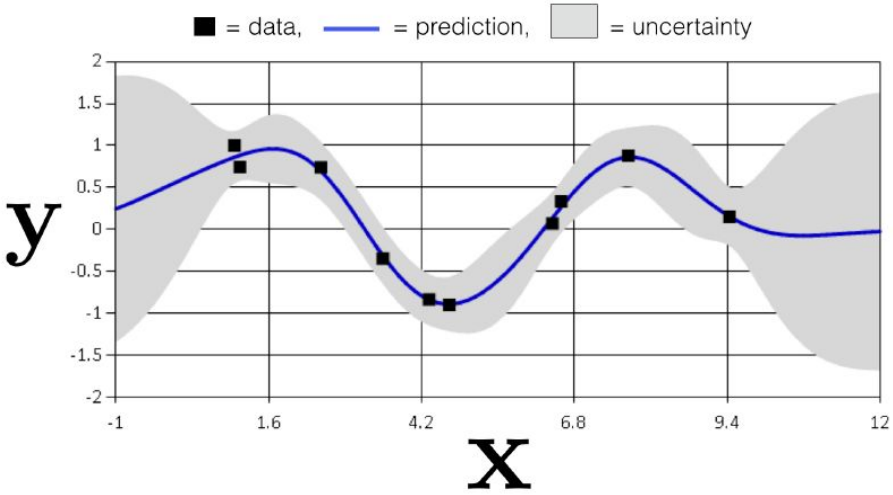




DeepMind

*equal contribution

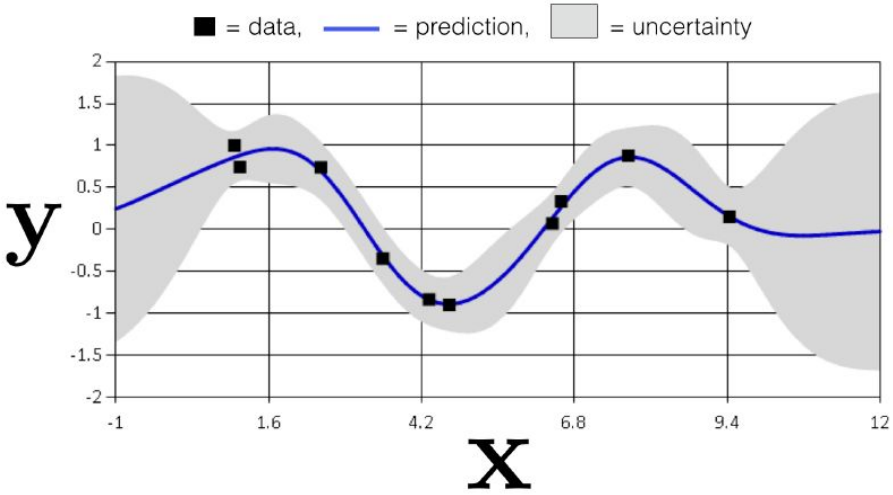
Predictive Models

$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$$



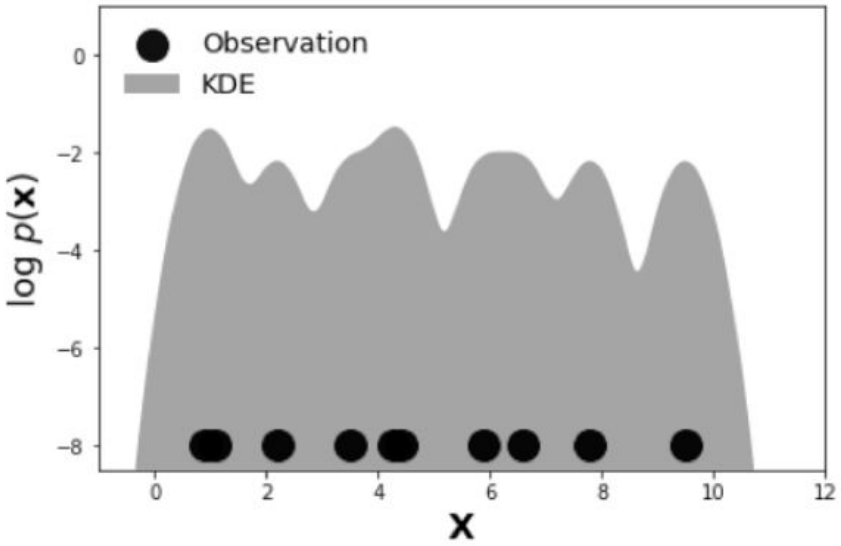
Predictive Models

$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$$



Generative Models

$$p(\mathbf{x};\boldsymbol{\phi})$$



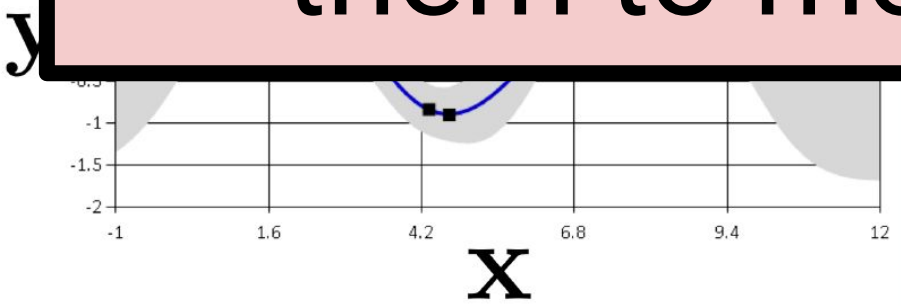
Predictive Models

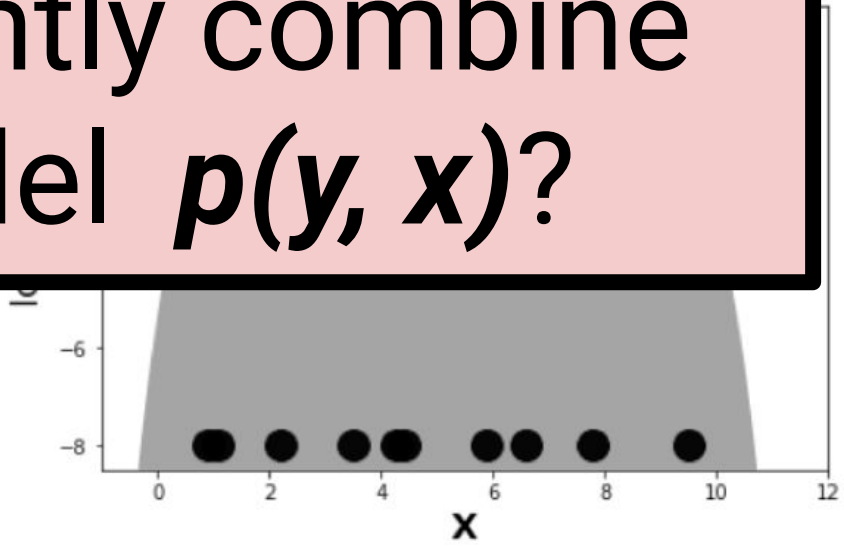
$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$$

Generative Models

$$p(\mathbf{x};\boldsymbol{\phi})$$

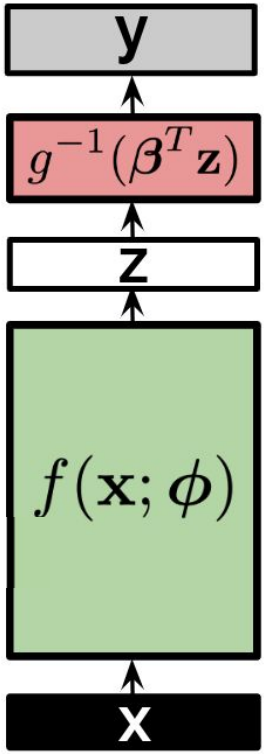
Can we efficiently combine them to model ***p(y, x)***?





Neural Hybrid Model

We define a computationally efficient **hybrid model** by combining *normalizing flows* with *generalized linear models* (GLMs).

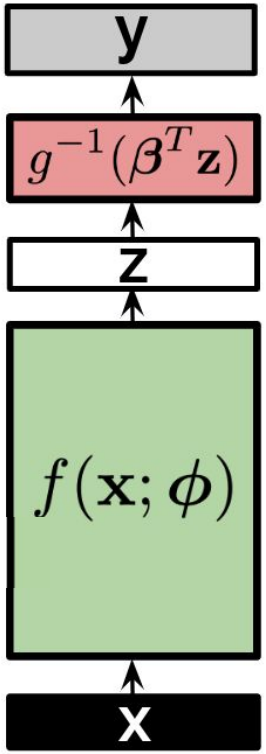


Deep Invertible Generalized Linear Model (DIGLM)

Neural Hybrid Model

We define a computationally efficient **hybrid model** by combining *normalizing flows* with *generalized linear models* (GLMs).

$$p(y_n, \boldsymbol{x}_n; \boldsymbol{\theta}) = \underbrace{p(y_n|\boldsymbol{x}_n; \boldsymbol{\beta}, \boldsymbol{\phi})}_{\text{Predictive Component}} \underbrace{p(\boldsymbol{x}_n; \boldsymbol{\phi})}_{\text{Generative Component}}$$



Deep Invertible Generalized Linear Model (DIGLM)

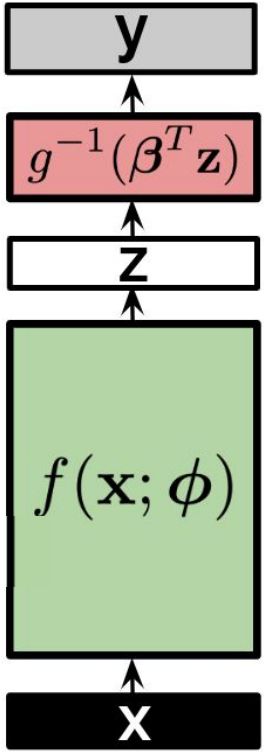
Neural Hybrid Model

We define a computationally efficient **hybrid model** by combining *normalizing flows* with *generalized linear models* (GLMs).

$$p(y_n, \boldsymbol{x}_n; \boldsymbol{\theta}) = p(y_n|\boldsymbol{x}_n; \boldsymbol{\beta}, \boldsymbol{\phi}) \; p(\boldsymbol{x}_n; \boldsymbol{\phi})$$

$$= \boxed{p(y_n|f(\boldsymbol{x}_n; \boldsymbol{\phi}); \boldsymbol{\beta})} \; \boxed{p_z(f(\boldsymbol{x}_n; \boldsymbol{\phi})) \left| \frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}}}{\partial \boldsymbol{x}_n} \right|}$$

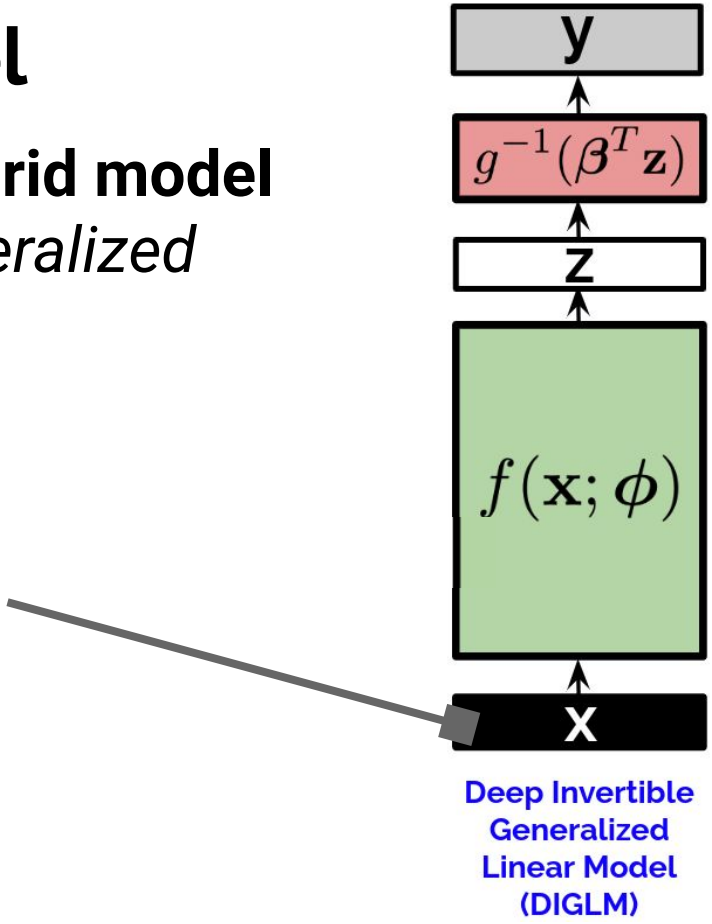
Linear Model *Normalizing Flow*



Deep Invertible Generalized Linear Model (DIGLM)

Neural Hybrid Model

We define a computationally efficient **hybrid model** by combining *normalizing flows* with *generalized linear models* (GLMs).

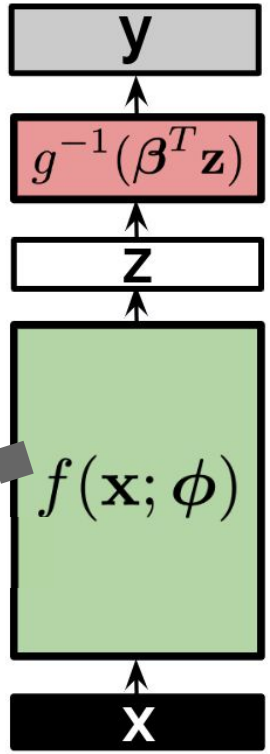


Input features.

Neural Hybrid Model

We define a computationally efficient **hybrid model** by combining *normalizing flows* with *generalized linear models* (GLMs).

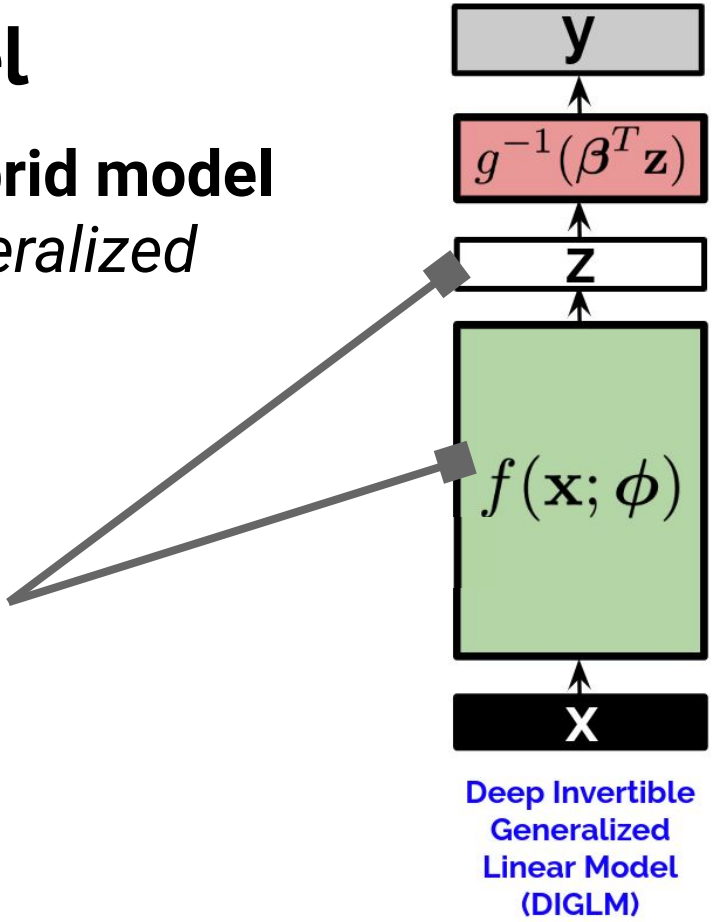
Normalizing flow acts as a deep neural feature extractor.



Neural Hybrid Model

We define a computationally efficient **hybrid model** by combining *normalizing flows* with *generalized linear models* (GLMs).

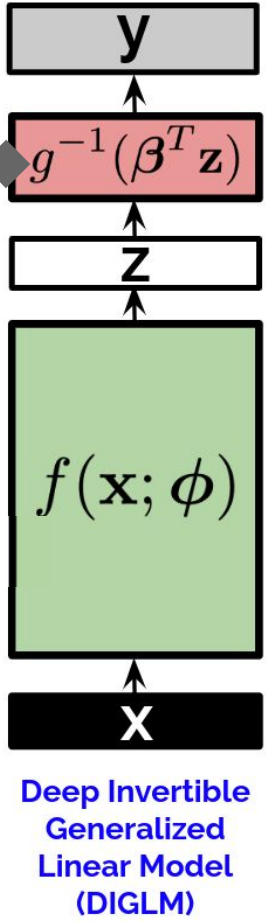
Flow's output and params. are used to compute ***p(x)*** *via change-of-variables.*



Neural Hybrid Model

We define a computationally efficient **hybrid model** by combining *normalizing flows* with *generalized linear models* (GLMs).

Flow's output is used as the feature vector in a (generalized) linear model, which computes ***p(y|x)***.



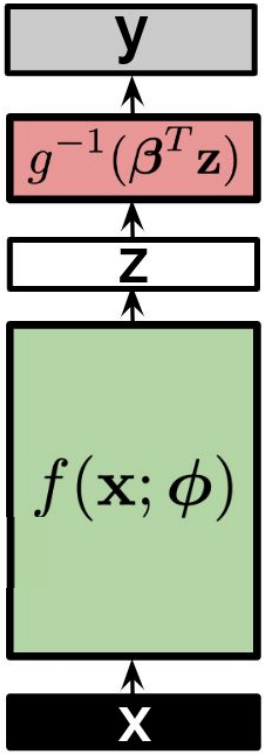
Neural Hybrid Model

We define a computationally efficient **hybrid model** by combining *normalizing flows* with *generalized linear models* (GLMs).

Weight to trade-off predictive and generative performance.

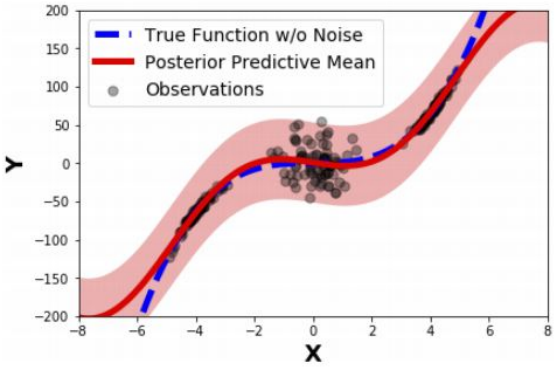
Optimization objective:

$$\mathcal{J}_\lambda(\boldsymbol{\theta}) = \sum_{n=1}^{N} \Big(\log p(y_n | \boldsymbol{x}_n; \boldsymbol{\beta}, \boldsymbol{\phi}) + \lambda \log p(\boldsymbol{x}_n; \boldsymbol{\phi}) \Big)$$



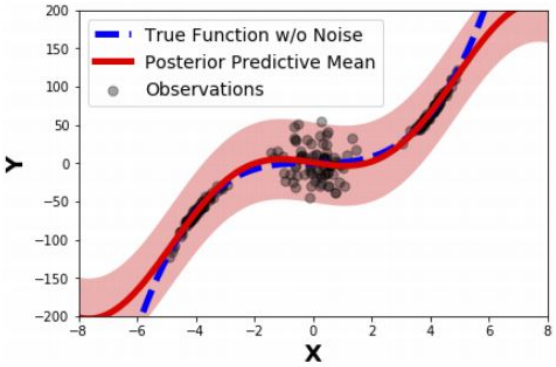
Deep Invertible Generalized Linear Model (DIGLM)

Simulation: Heteroscedastic Regression

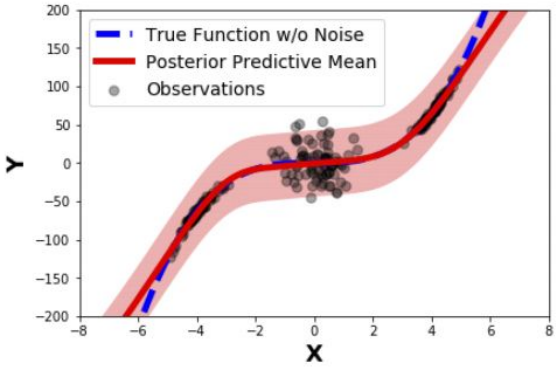


Gaussian process fitted to simulated data.

Simulation: Heteroscedastic Regression

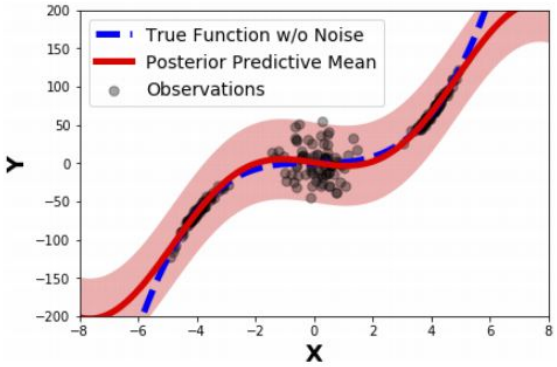


Gaussian process fitted to simulated data.

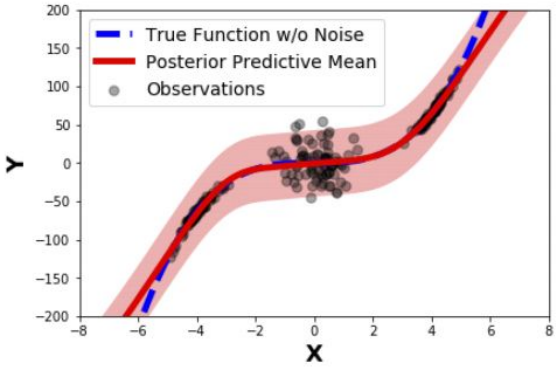


Our model's predictive component.

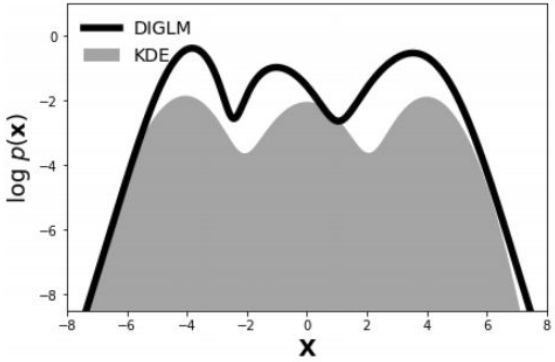
Simulation: Heteroscedastic Regression



Gaussian process fitted to simulated data.



Our model's predictive component.



Our model's generative component.

For more details, please visit our poster.

Hybrid Models with Deep and Invertible Features

Eric Nalisnick*, Akihiro Matsukawa*, Yee Whye Teh, Dilan Gorur, Balaji Lakshminarayanan

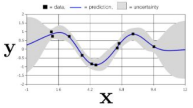
DeepMind

*equal contribution

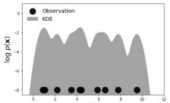
1. Introduction

- Neural networks usually model the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$, where $\boldsymbol{y}$ denotes a label and $\boldsymbol{x}$ features.
- Generative models, on the other hand, represent the distribution over features $p(\boldsymbol{x})$.
- **Can we efficiently combine the two in a hybrid model of the joint distribution $p(\boldsymbol{y}, \boldsymbol{x})$?**

Conditional Model



Generative Model



2. Background

Invertible Generative Models (Normalizing Flows)

Invertible generative models (a.k.a. normalizing flows) are a broad class of models defined via the change-of-variables formula. An initial density **p(x)** 'flows' through a series of transformations **f(x)** and morphs into some (usually simpler) prior distribution **p(z)**.

$$\log p_x(\boldsymbol{x}) = \log p_z(f(\boldsymbol{x};\boldsymbol{\phi})) + \log \left| \frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}}}{\partial \boldsymbol{x}} \right|$$

Generalized Linear Models (GLMs)

Generalized linear models (GLMs) model the expected response (or label) $\mathbf{y}$ as a transformation of the linear model $\boldsymbol{\beta}^T\mathbf{z}$ where $\boldsymbol{\beta}$ are parameters and $\mathbf{z}$ are features (covariates).

$$\mathbb{E}[y_n|\boldsymbol{z}_n] = g^{-1}\left(\boldsymbol{\beta}^T \boldsymbol{z}_n\right)$$

- **Regression**: $\mathbb{E}[y|\mathbf{z}] = \texttt{identity}(\boldsymbol{\beta}^T\mathbf{z})$
- **Binary Classification**: $\mathbb{E}[y|\mathbf{z}] = \texttt{logistic}(\boldsymbol{\beta}^T\mathbf{z})$

3. Combining Deep Generative Models and Linear Models

We define a model of the joint distribution $p(y, x)$ by instantiating a GLM on the output of a normalizing flow:

$$p(y_n, \boldsymbol{x}_n; \boldsymbol{\theta}) = p(y_n|\boldsymbol{x}_n; \boldsymbol{\beta}, \boldsymbol{\phi})\ p(\boldsymbol{x}_n; \boldsymbol{\phi})$$
$$= p(y_n|f(\boldsymbol{x}_n;\boldsymbol{\phi});\boldsymbol{\beta})\ p_z(f(\boldsymbol{x}_n;\boldsymbol{\phi}))\ \left|\frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}}}{\partial \boldsymbol{x}_n}\right|$$

In practice, we add a weight to the flow terms to tradeoff between predictive and generative behavior:

$$\mathcal{J}_\lambda(\boldsymbol{\theta}) = \sum_{n=1}^{N}\Big(\log p(y_n|\boldsymbol{x}_n;\boldsymbol{\beta},\boldsymbol{\phi}) + \lambda \log p(\boldsymbol{x}_n;\boldsymbol{\phi})\Big)$$

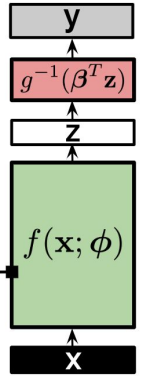
$\log \lvert\partial \mathbf{f}/\partial \mathbf{x}\rvert$	Examples
Planar:	$= \lvert 1 + \mathbf{u}^T f'(\mathbf{w}^T\mathbf{x} + b)\mathbf{w}\rvert$ where $\boldsymbol{w}, \boldsymbol{u}$ are parameters.
RNVP:	$= \sum_l \sum_d s_{l,d}(\mathbf{x};\boldsymbol{\phi})$ where $\boldsymbol{s()}$ are scaling operations.
Glow:	$= \sum_l \sum_d s_{l,d}(\mathbf{x};\boldsymbol{\phi}) + h_l w_l \log \lvert\det \mathbf{W}_1\rvert$, $\boldsymbol{W}$ 1x1 params.

Bayesian treatment: we can place a prior on the parameters of the GLM in order to quantify model *and* data uncertainty.

$$f(\boldsymbol{x};\boldsymbol{\phi}) \sim p(\boldsymbol{z}),\ \ \boldsymbol{\beta} \sim p(\boldsymbol{\beta}),\ \ y_n \sim p(y_n|f(\boldsymbol{x}_n;\boldsymbol{\phi}),\boldsymbol{\beta})$$

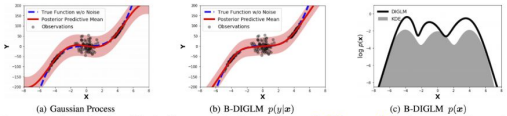
For a Gaussian prior on the GLM, the predictive model can be trained via the closed-form marginal likelihood:

$$\log p(y_n|f(\boldsymbol{x}_n;\boldsymbol{\phi})) = \log \mathrm{N}\left(\boldsymbol{y};\ \mathbf{0},\ \sigma_0^2\mathbb{I} + \lambda^{-1}\boldsymbol{Z}_{\boldsymbol{\phi}}\boldsymbol{Z}_{\boldsymbol{\phi}}^T\right)$$



Deep Invertible Generalized Linear Model (DIGLM)

4. Simulation

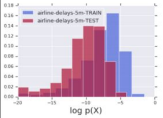


(a) Gaussian Process (b) B-DIGLM $p(y|\boldsymbol{x})$ (c) B-DIGLM $p(\boldsymbol{x})$

1D regression task with heteroscedastic noise. **Subfigure (a)** shows a Gaussian process and **Subfigure (b)** shows our Bayesian DIGLM. **Subfigure (c)** shows **$p(\boldsymbol{x})$** learned by the same DIGLM (black line) and compares it to a KDE (gray shading).

5. Experiments

Regression on Flight Delay Data Set (N=5 million, D=8)



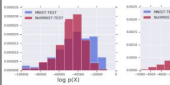
Model	RMSE ↓	NLL ↓
Mondrian Forests (SOTA)	**38.38**	6.91
DIGLM	40.46	**5.07**

- **This data set exhibits covariate shift** between the train and test splits.
- The DIGLM's **$p(\boldsymbol{x})$** component is able to detect this shift (see left).

Classification on MNIST and SVHN

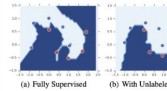
Model	MNIST BPD ↓	MNIST error ↓	MNIST NLL ↓	NotMNIST BPD ↑	NotMNIST NLL ↓	NotMNIST Entropy ↑
Discriminative ($\lambda = 0$)	81.80*	**0.67%**	0.062	87.74*	29.27	0.130
Hybrid ($\lambda = 0.01/D$)	1.83	0.75%	**0.035**	5.84	2.36	**2.300**
Hybrid ($\lambda = 1.0/D$)	1.26	2.22%	0.081	6.13	**2.30**	**2.300**
Hybrid ($\lambda = 10.0/D$)	**1.25**	4.01%	0.145	**6.17**	**2.30**	**2.300**

Model	SVHN BPD ↓	SVHN error ↓	SVHN NLL ↓	CIFAR-10 BPD ↑	CIFAR-10 NLL ↓	CIFAR-10 Entropy ↑
Discriminative ($\lambda = 0$)	15.40*	**4.26%**	**0.225**	15.20*	4.60	0.998
Hybrid ($\lambda = 0.1/D$)	3.35	4.86%	0.260	7.06	5.06	1.153
Hybrid ($\lambda = 1.0/D$)	2.40	5.23%	0.253	6.16	4.23	1.677
Hybrid ($\lambda = 10.0/D$)	**2.23**	7.27%	0.268	**7.03**	**3.69**	**2.143**



- λ controls the trade-off between p(y|x) and p(x).
- **Hybrid model is better able to detect the OOD inputs via p(x).**

Semi-Supervised Learning: MNIST and Half Moons



Half-moons simulation: the DIGLM leverages unlabeled data to learn a smooth decision boundary (N=10 labeled points).

Model	MNIST-error ↓	MNIST-NLL ↓
1000 labels only	6.61%	0.276
1000 labels + unlabeled	0.99%	0.069
All labeled	**0.73%**	**0.035**

SSL (VAT) with only 1000 labels (2% of labeled data) achieves <1% error on MNIST

6. Summary

We defined a neural hybrid model that can efficiently compute both predictive p(y|x) and generative p(x) distributions, in a single feed-forward pass, making it a useful building block for downstream applications of probabilistic deep learning.

Paper: https://arxiv.org/abs/1902.02767