Hybrid Models with Deep and Invertible Features

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

*equal contribution
Predictive Models

\[ p(y \mid x; \theta) \]
**Predictive Models**

\[ p(y|x; \theta) \]

**Generative Models**

\[ p(x; \phi) \]
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).
We define a computationally efficient \textbf{hybrid model} by combining \textit{normalizing flows} with \textit{generalized linear models} (GLMs).

\[
p(y_n, x_n; \theta) = p(y_n | x_n; \beta, \phi) \ p(x_n; \phi)
\]
Neural Hybrid Model

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

\[
p(y_n, x_n; \theta) = p(y_n | x_n; \beta, \phi) \ p(x_n; \phi) = p(y_n | f(x_n; \phi); \beta) p_z(f(x_n; \phi)) \ \frac{\partial f_\phi}{\partial x_n}
\]

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

**Optimization objective:**

\[
J_\lambda(\theta) = \sum_{n=1}^{N} \left( \log p(y_n|x_n; \beta, \phi) + \lambda \log p(x_n; \phi) \right)
\]
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

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

**2. Background**

*Invertible Generative Models (Normalizing Flows)*

Invertible generative models (aka, normalizing flows) are a broad class of models defined via the change-of-variables formula. An initial density $p_{0}(x)$ flows through a series of transformations $f_{l}$ and morphs into some usually simpler prior distribution $p_{2}$.

$$
\log p_{2}(x) = \log p_{0}(f(x; \phi)) + \log \frac{\partial f_{\phi}}{\partial x}
$$

*Generalized Linear Models (GLMs)*

Generalized linear models (GLMs) model the expected response (or label) $y$ as a transformation of the linear model $g^{-1}(\beta_{1}x)$. We use parameters $\beta$ and features $x$.

$$
E[y|x] = g^{-1}(\beta_{1}x)
$$

- Regression: $E[y|x] = \text{identity}(\beta_{1}x)$
- Binary Classification: $E[y|x] = \text{logistic}(\beta_{1}x)$

**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:

$$
p(y,x; \theta) = p(y|x; \beta, \phi) p(x; \phi)
$$

$$
p(y|x; f(x; \phi); \beta) = p_{f}(f(x; \phi)) \frac{\partial f_{\phi}}{\partial x}
$$

In practice, we add a weight to the flow terms to trade off between predictive and generative behavior.

$$
J_{\phi}(\theta) = \sum_{n=1}^{N} \log p(y|x; \beta, \phi) + \lambda \log p(x; \phi)
$$

**4. Simulation**

- 1D regression task with heteroscedastic noise. Subfigure (a) shows a Gaussian process and Subfigure (b) shows our Bayesian DGLM. Subfigure (c) shows p(d) learned by the same DGLM (black line) and compares it to a KDE (gray shading).

**5. Experiments**

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

- This data set exhibits covariate shift between the train and test splits.
- The DGLM’s p(d) component is able to detect this shift (see left).

Classification on MNIST and SVHN:

- 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 DGLM leverages unlabeled data to learn a smooth decision boundary (N=0 labeled points).

**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.