Actuarial Neural Networks and Uncertainty

AFRIC 2

Patrick Laub, UNSW

Joint work with Benjamin Avanzi, **Eric Dong**, and Bernard Wong

Distributional Regression

Why avoid neural networks

They're not *inherently interpretable*, so just have to look at inputs and outputs from the black box

Why avoid neural networks

They're not *inherently interpretable*, so just have to look at inputs and outputs from the black box

NN models typically just output a prediction without any sense of its *confidence*

Why avoid neural networks

They're not *inherently interpretable*, so just have to look at inputs and outputs from the black box

NN models typically just output a prediction without any sense of its *confidence*

$\therefore$ Cannot *trust* the NN models. **Dealbreaker**

Why avoid neural networks

They're not *inherently interpretable*, so just have to look at inputs and outputs from the black box

NN models typically just output a prediction without any sense of its *confidence*

∴ Cannot *trust* the NN models. **Dealbreaker**

Now let's focus on actuarial problems.

Car insurance

Claim size prediction

👤 **Age**	🚗 **Age**	🏎️ **Type**
25	3	🚙 Sedan
40	5	🚐 SUV
19	1	🏎️ Sports Car
60	10	🚘 Hatchback

Car insurance

Claim size prediction

👤 **Age**	🚗 **Age**	🏎️ **Type**
25	3	🚙 Sedan
40	5	🚐 SUV
19	1	🏎️ Sports Car
60	10	🚘 Hatchback

$\longrightarrow$

Car insurance

Claim size prediction

👤 Age	🚗 Age	🏎️ Type		Cost
25	3	🚙 Sedan		💵 $1,200
40	5	🚐 SUV	→	💵 $2,500
19	1	🏎️ Sports Car		💵 $3,800
60	10	🚘 Hatchback		💵 $800

Car insurance

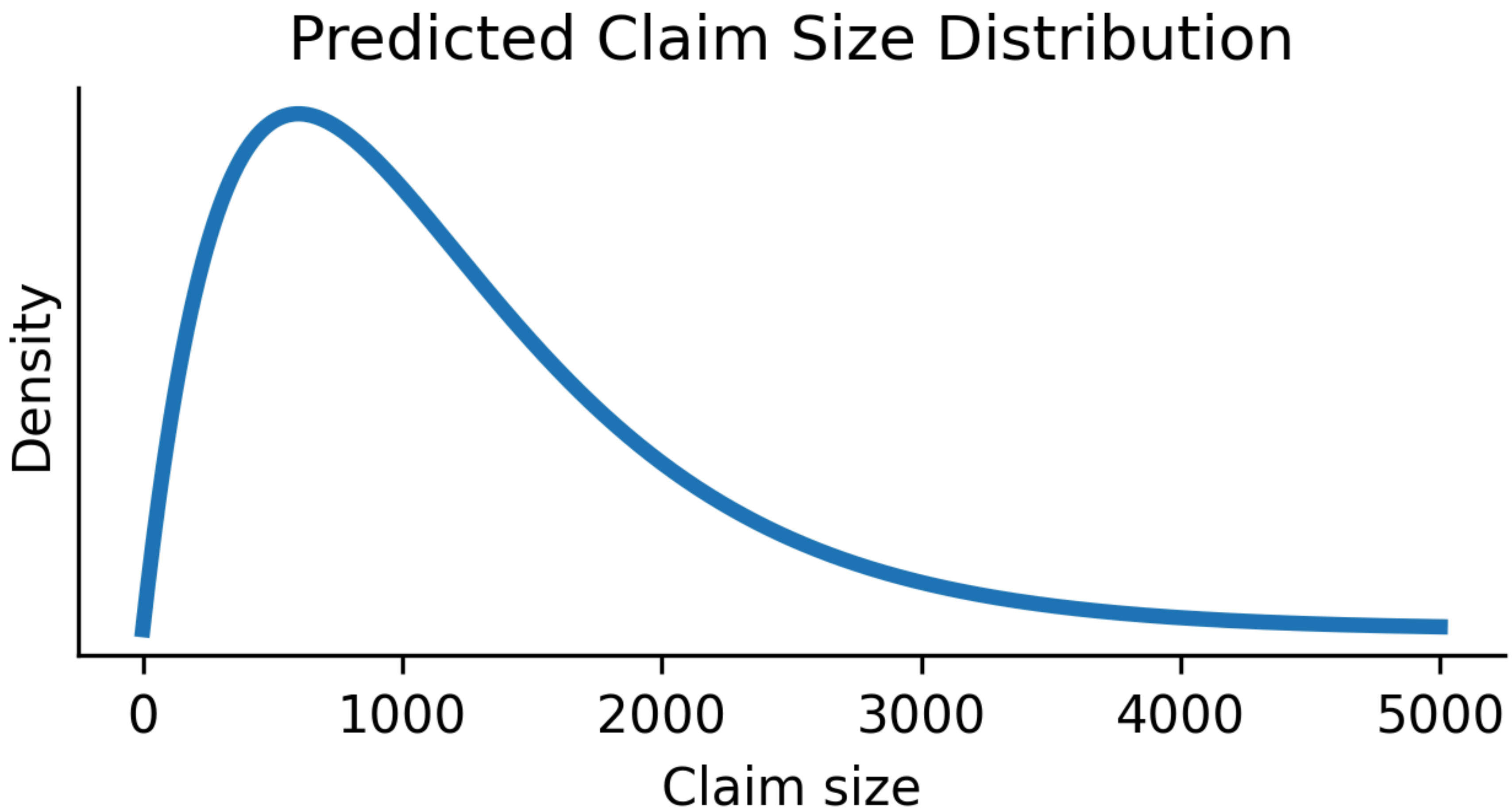
Claim size prediction

👤 **Age**	🚗 **Age**	🏎️ **Type**		**Cost**
25	3	🚙 Sedan		💵 $1,200
40	5	🚐 SUV	$\longrightarrow$	💵 $2,500
19	1	🏎️ Sports Car		💵 $3,800
60	10	🚘 Hatchback		💵 $800

What's wrong? Not enough rows? Not enough columns?

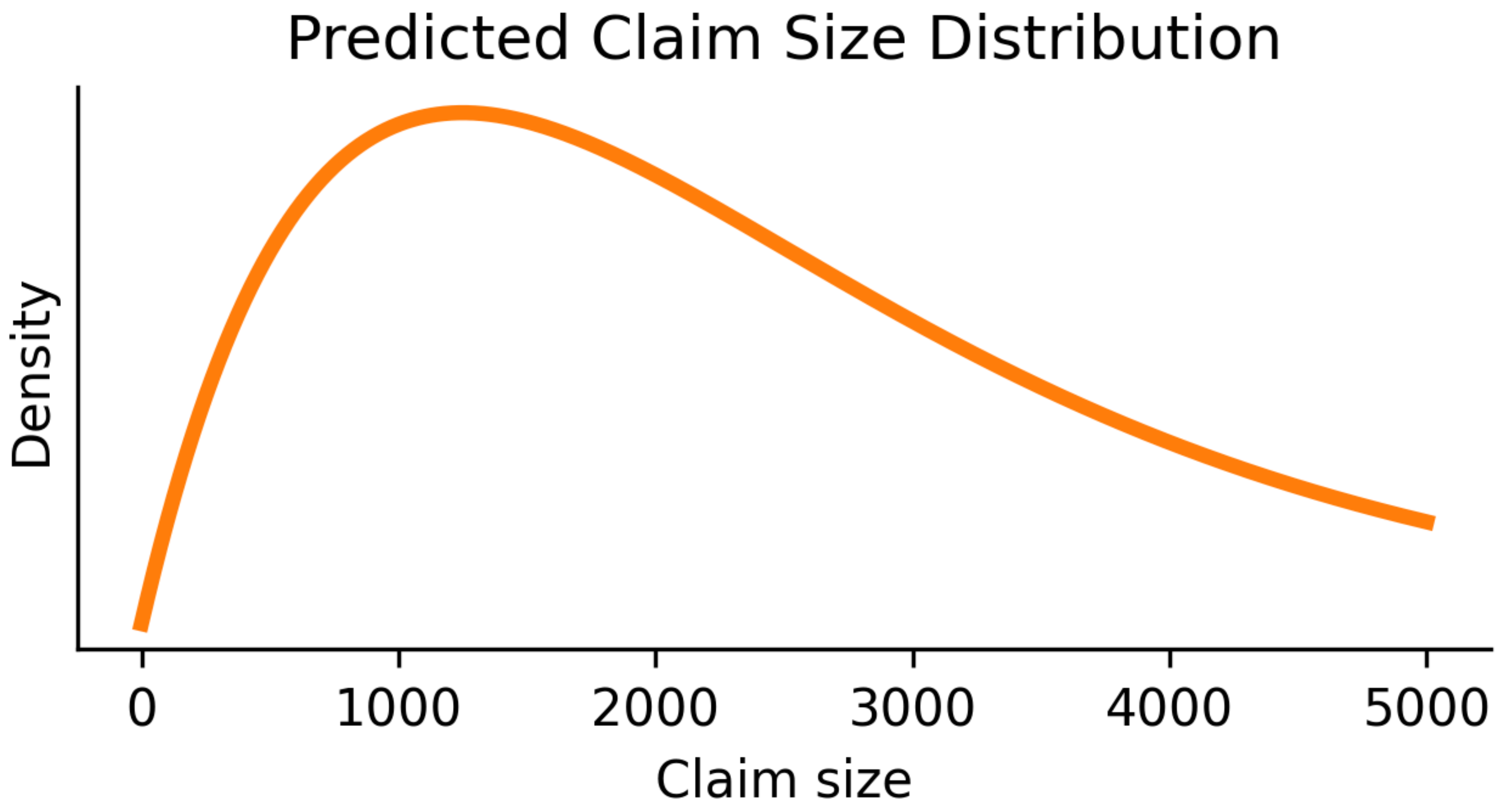
Distributional regression

Customer 1 = (25, 3, 🚙)



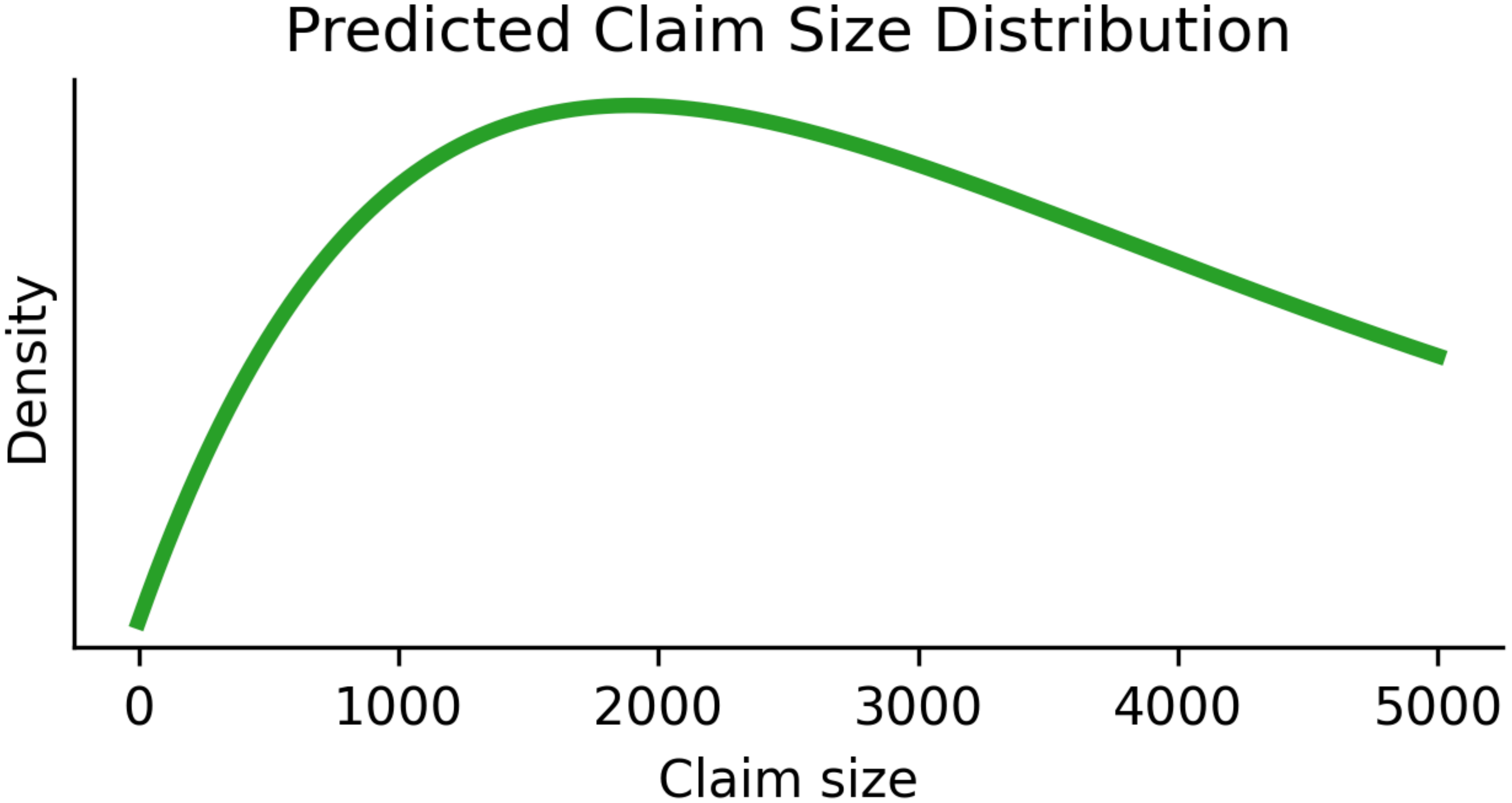
Distributional regression

Customer 2 = (40, 5, 🚐)



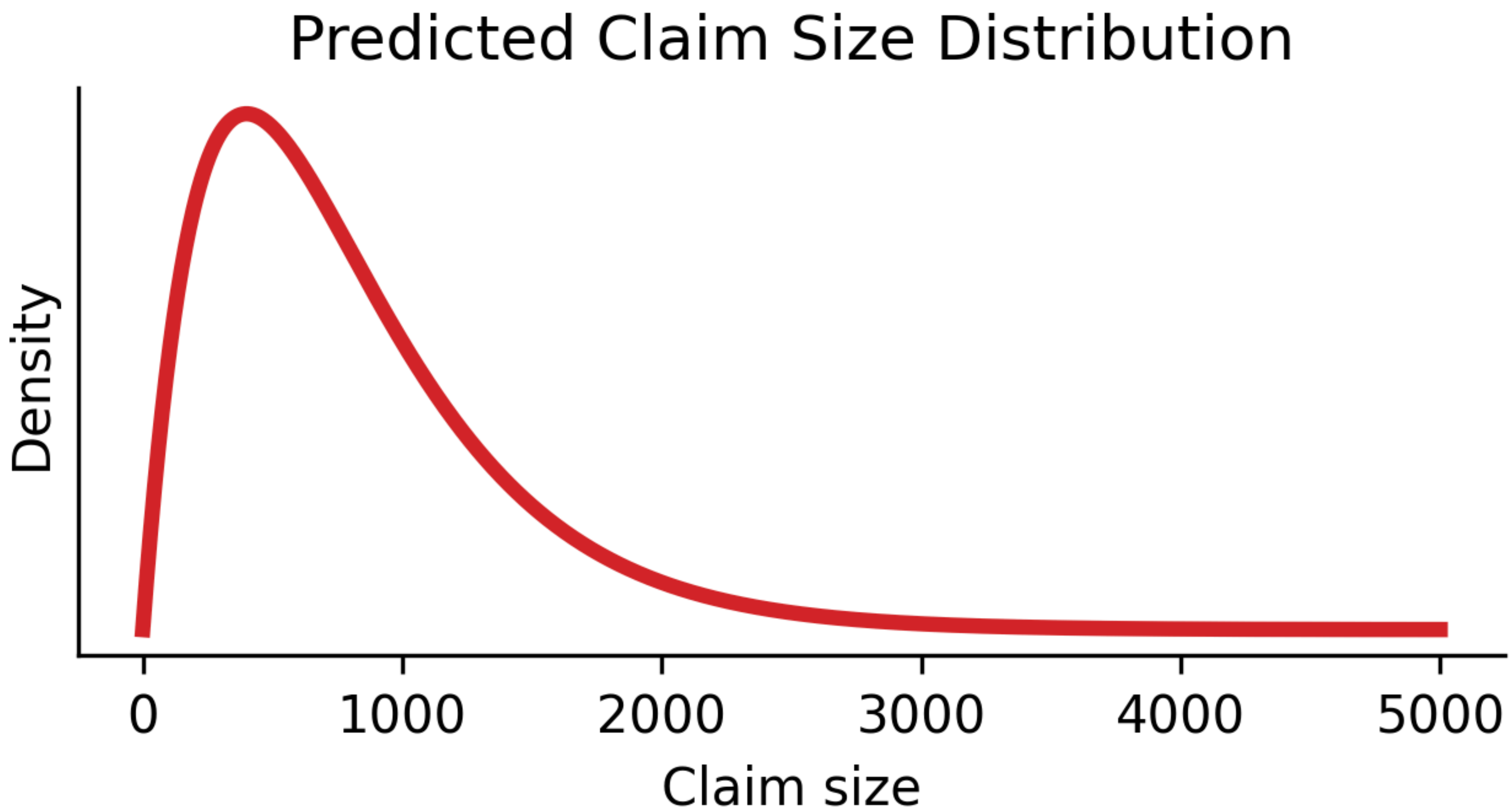
Distributional regression

Customer 3 = (19, 1, 🏎️)



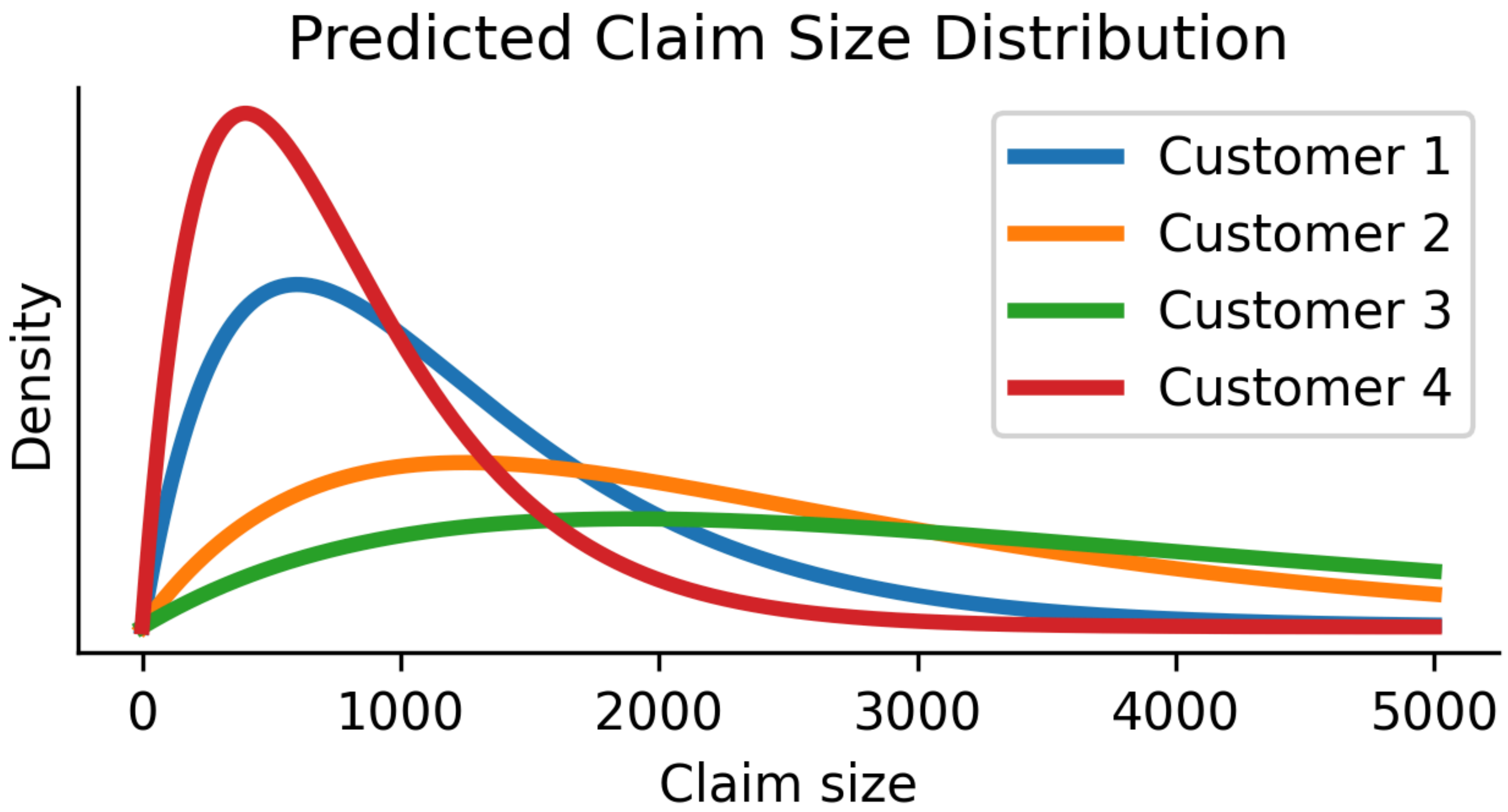
Distributional regression

Customer 4 = (60, 10, 🚘)



Distributional regression

All customers



Current solutions

UNSW SYDNEY

A generalised linear model

A gamma GLM with a log link function:

$$
\begin{aligned}
Y|\mathbf{X} &\sim \text{Gamma}(\ldots,\ldots) \\
\mathbb{E}[Y|\mathbf{X}] &= \exp\Big\{\beta_0 + \beta_1 \cdot \text{Age} + \beta_2 \cdot \text{Car Age} + \beta_3 \cdot \text{Type}\Big\}
\end{aligned}
$$

A simple model, easy to train and interpret, but…

A generalised linear model

A gamma GLM with a log link function:

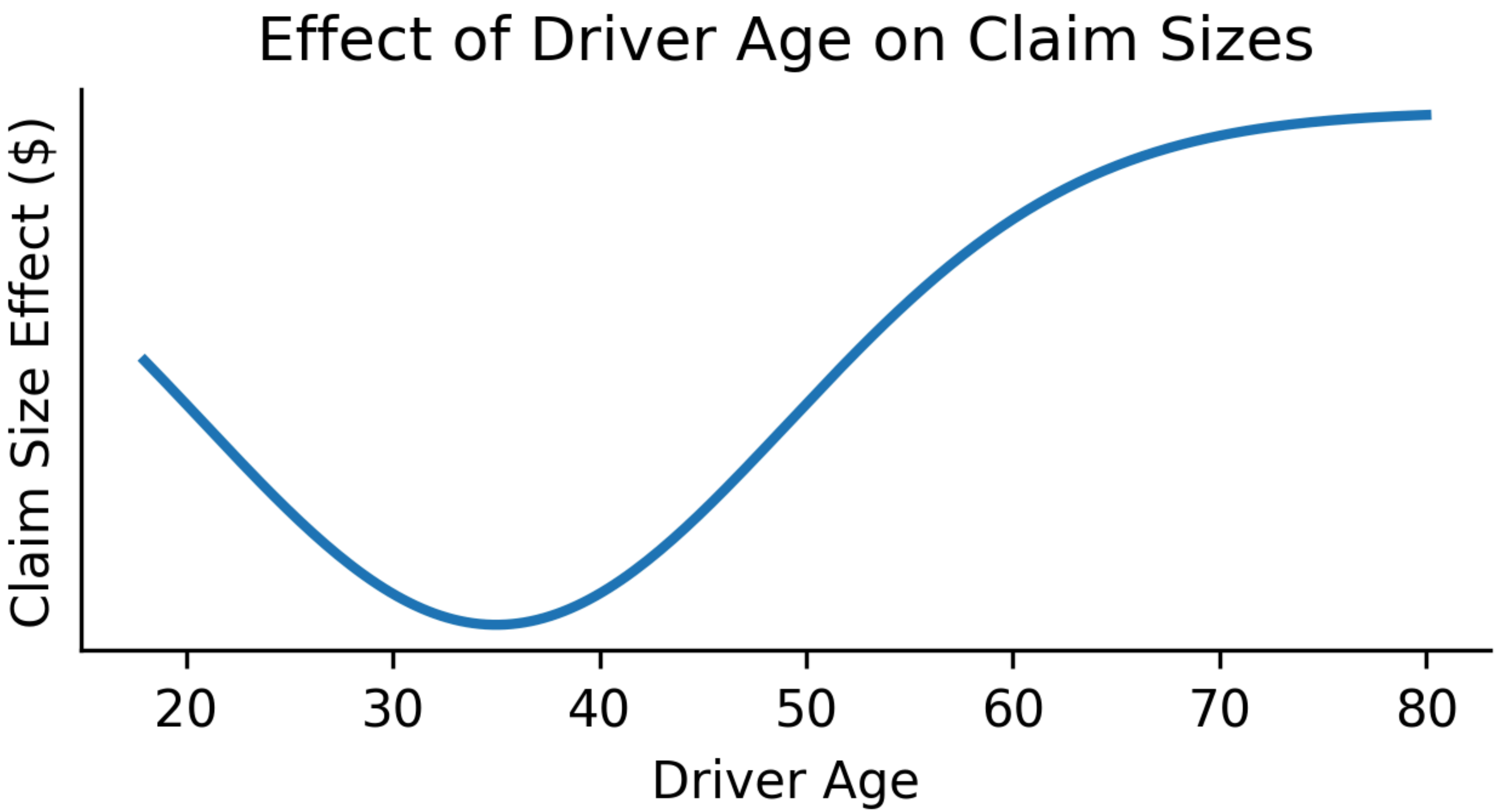
$$
\begin{aligned}
Y \,|\, \mathbf{X} &\sim \mathrm{Gamma}(\ldots, \ldots) \\
\mathbb{E}[Y \,|\, \mathbf{X}] &= \exp\Big\{ \beta_0 + \beta_1 \cdot \mathrm{Age} + \beta_2 \cdot \mathrm{Car\ Age} + \beta_3 \cdot \mathrm{Type} \Big\}
\end{aligned}
$$

A simple model, easy to train and interpret, but…

GLMs can be

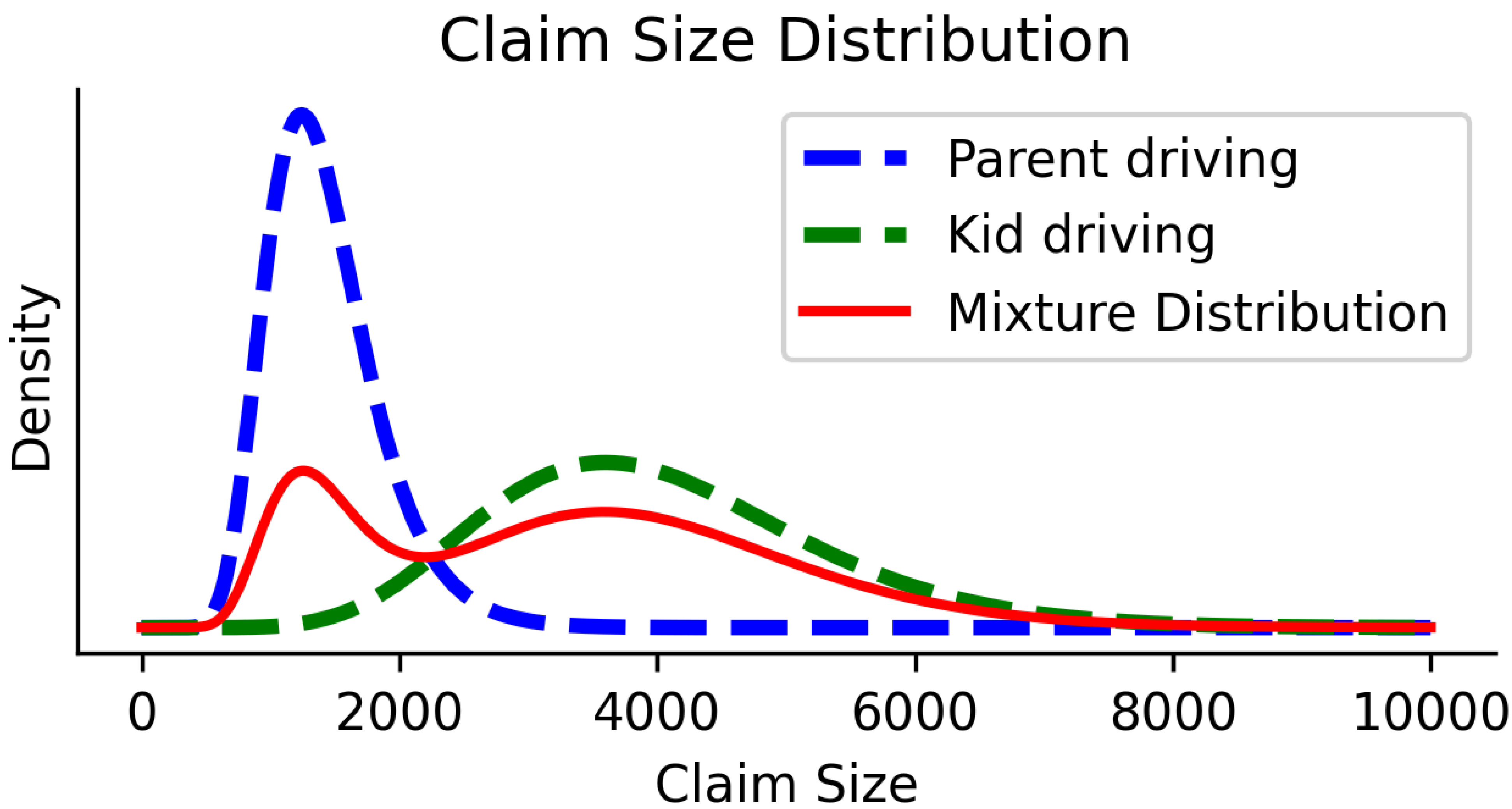
1. Bad at *regression*
2. Bad at *distributional* regression

Example 1: Non-monotonicity



GLMs cannot (easily) do this $\longrightarrow$ Use a neural network

Example 2: Multi-modality



CANN

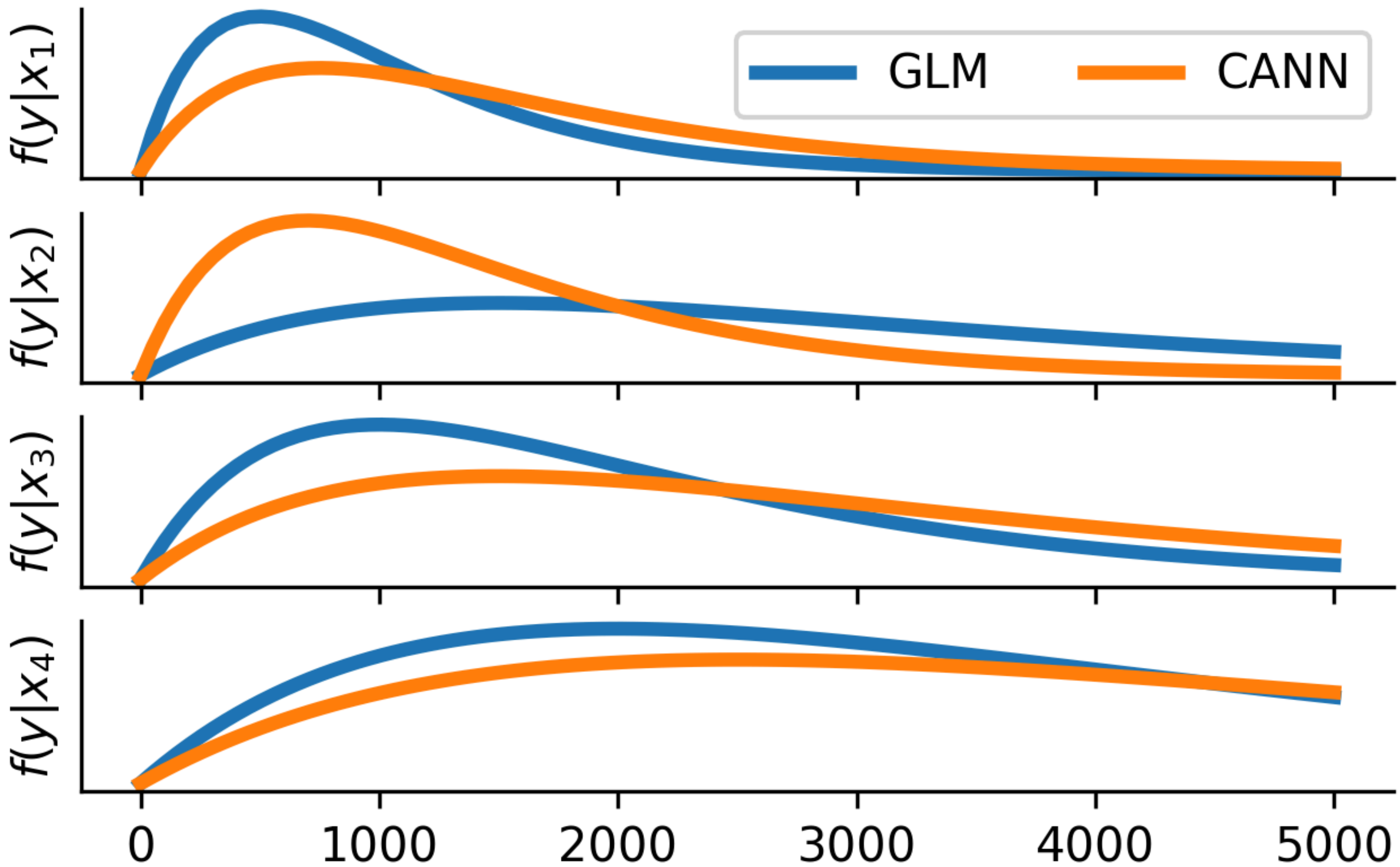
"Combined Actuarial Neural Network" by Schelldorfer and Wüthrich (2019):

1. Fit a GLM with $\boldsymbol{\beta}$ and link function $g(\cdot)$
2. Fit a neural network $\mathcal{M}_{\text{CANN}}$ that predicts

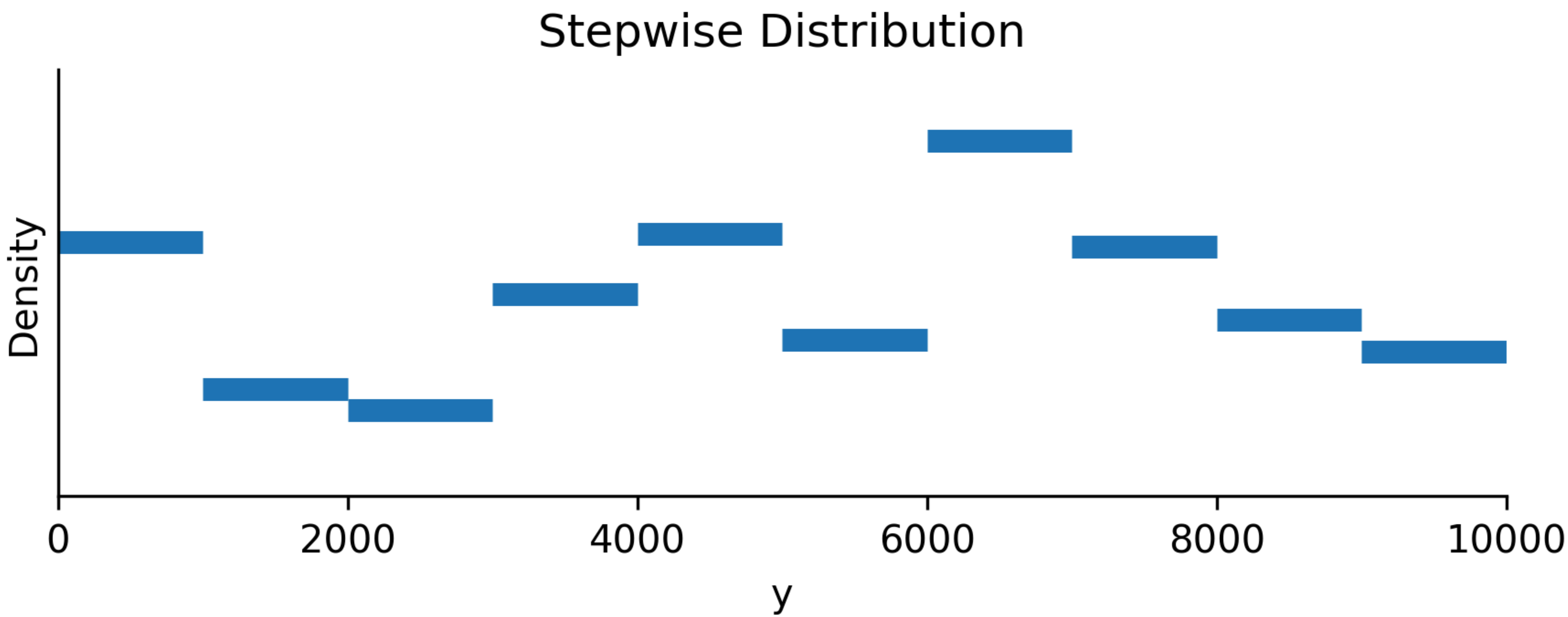
$$\mathbb{E}[Y|\boldsymbol{X}=\boldsymbol{x}] = g^{-1}\Big(\langle \boldsymbol{\beta}, \boldsymbol{x}\rangle + \mathcal{M}_{\text{CANN}}(\boldsymbol{x}; \boldsymbol{w}_{\text{CANN}})\Big).$$

This makes the 'regression' part smarter, but not the 'distribution' part

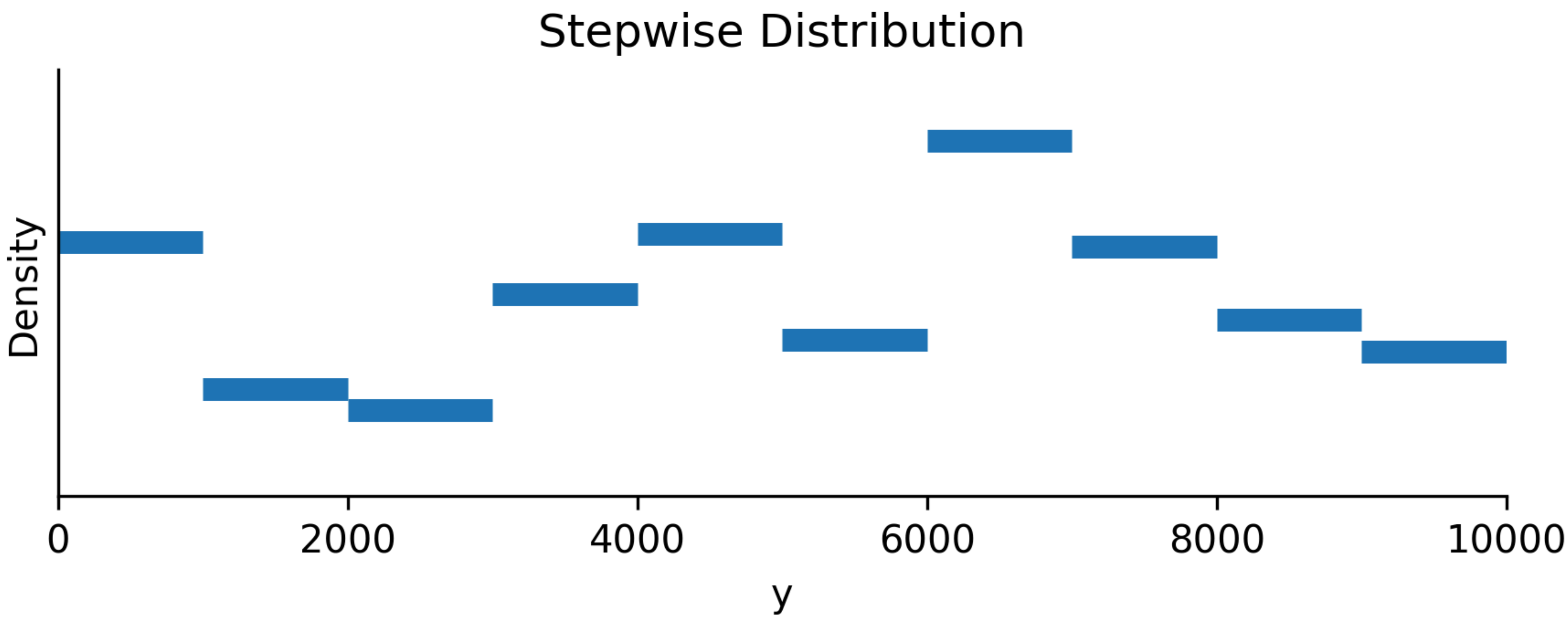
Shifting the predicted distributions



Deep Distributional Regression



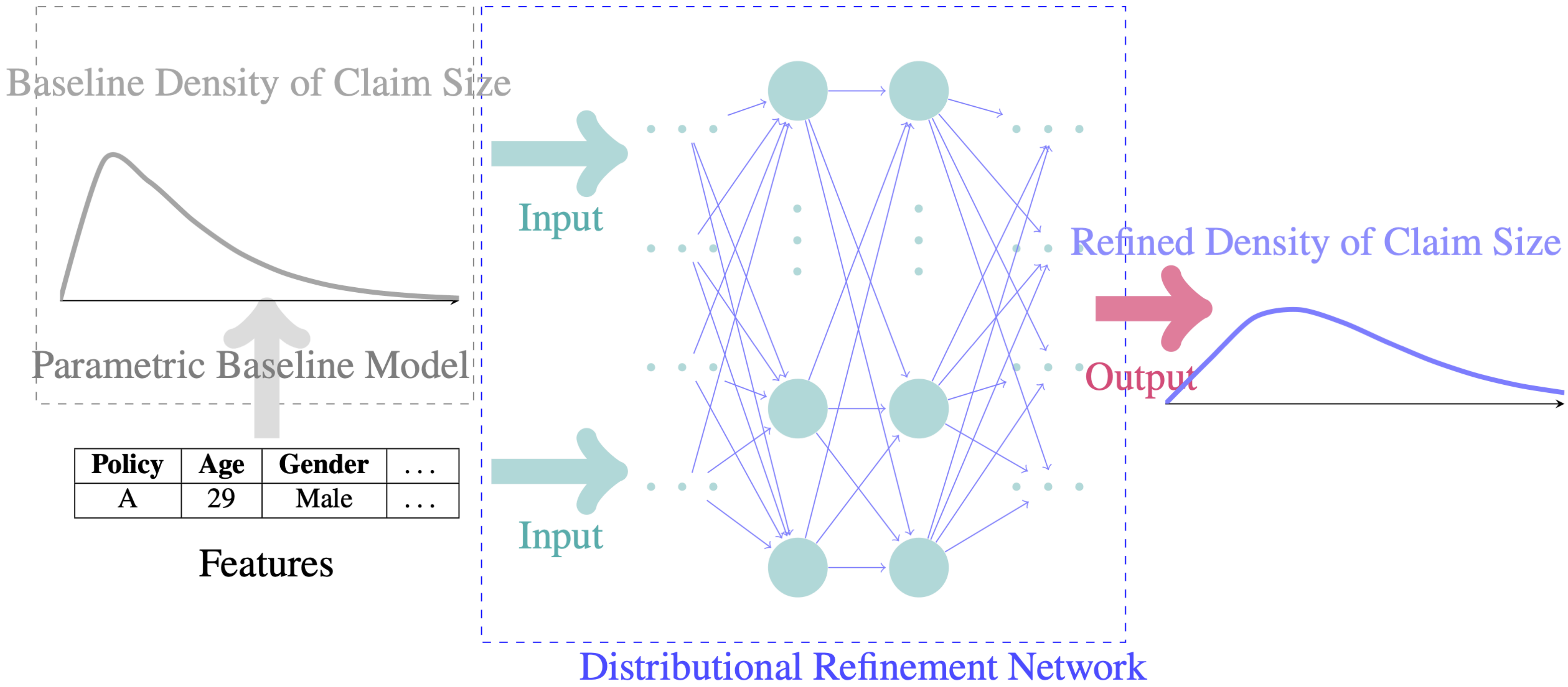
Deep Distributional Regression



More flexible regression (NN), and most flexible distributional outputs (non-parametric), **uninterpretable**

Distributional Refinement Network

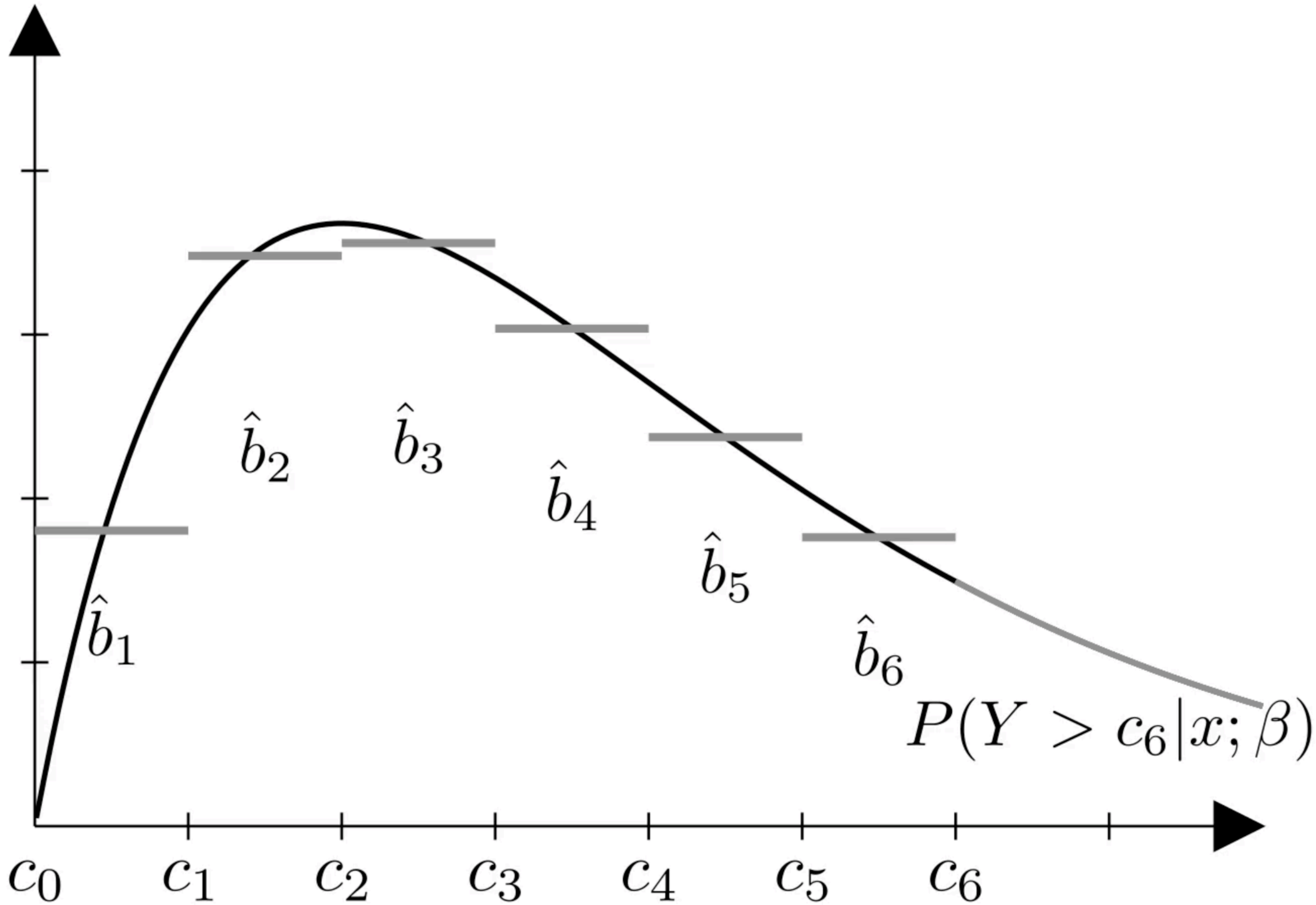
Distributional Refinement Network



DRN first uses a trusted baseline model, then makes small adjustments to it

Baselines

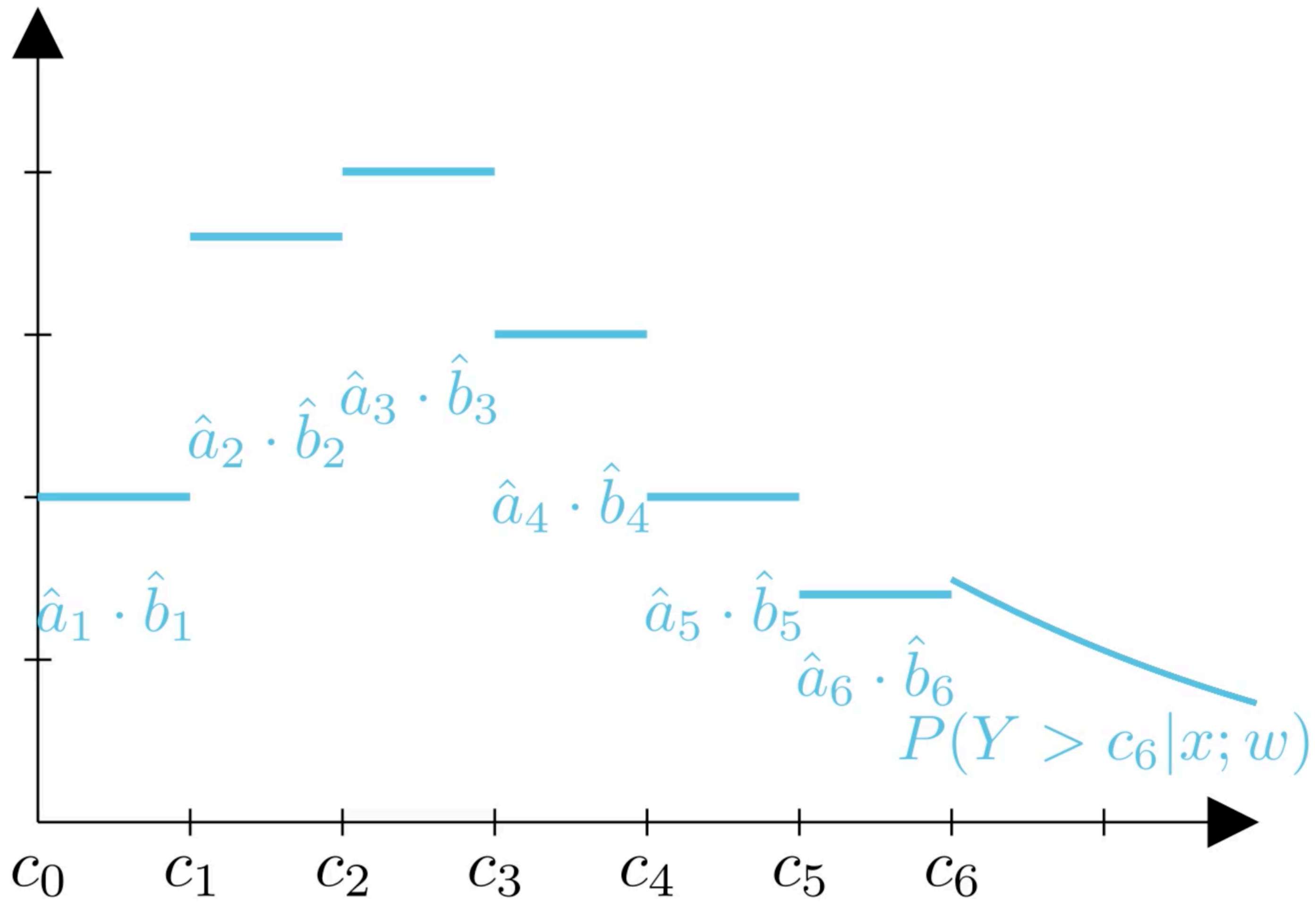
Add in a baseline model, and "discretise" it



Baseline Probability Masses

Adjustments

Then adjust the heights of the bins with NN:



DRN output distribution

Loss and Regularisation

We minimise

$$\begin{aligned}\text{Loss} = {} & \text{Distributional Accuracy (e.g. NLL)} \\ & + \text{Baseline Resemblance (e.g. KL Div.)} \\ & + \text{Density Smoothness (e.g. Second-Order Difference)}\end{aligned}$$

Loss and Regularisation

We minimise

$$\begin{aligned}\text{Loss} = &\ \text{Distributional Accuracy (e.g. NLL)} \\ &+ \text{Baseline Resemblance (e.g. KL Div.)} \\ &+ \text{Density Smoothness (e.g. Second-Order Difference)}\end{aligned}$$

So we have a *lever* to control how far the NN can deviate from a *trusted inherently interpretable baseline*.

This is used in ChatGPT

State-of-the-art LLMs still use this approach

OpenAI

"Following Jaques et al. (2017; 2019), we use a KL constraint to prevent the fine-tuned model from drifting too far from the pretrained model."

- *Fine-Tuning Language Models from Human Preferences* (2019)
- *Learning to summarize from human feedback* (2020)
- *InstructGPT* / *ChatGPT* (2022)
- *Direct Preference Optimization (DPO)* (2024)

Google deepseek

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$
$$\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i, \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\varepsilon, 1+\varepsilon\right)A_i\right) - \beta\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right)\right), \quad (1)$$

$$\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right) = \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - \log\frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - 1, \quad (2)$$

Social RL - Natasha Jaques 35

The KL term in AI chat models

UNSW SYDNEY

Code: Training DRN

Loaded the datasets already, then make the cutpoints:

```
from drn import GLM, DRN, drn_cutpoints, train

left = 0
right = Y_train.max().item() * 1.1

cutpoints = drn_cutpoints(left, right, proportion=0.1, y=y_train)
print(cutpoints)
```

```
[0.00, 1.36, 2.72, 4.09, 5.45, 6.81, 8.17, 9.53, 10.90, 12.26, 13.62, 14.98,
16.34, 17.70, 19.07, 20.43, 21.79, 23.15, 24.51, 25.88, 27.24, 29.96, 32.69,
55.84, 132.10, 155.26, 179.77]
```

Then train the GLM and the DRN:

```
glm_model = GLM.from_statsmodels(X_train, Y_train, distribution="gamma")

drn_model = DRN(glm_model, cutpoints, hidden_size=256, num_hidden_layers=2)

train(drn_model, train_dataset, val_dataset, epochs=100, patience=5)
```

Code: Training CANN, MDN, DDR

```
from drn import CANN, MDN, DDR

cann_model = CANN(glm_model)
train(cann_model, train_dataset, val_dataset, epochs=100, patience=5)
cann_model.update_dispersion(X_train, Y_train)

mdn_model = MDN(X_train.shape[1])
train(mdn_model, train_dataset, val_dataset, epochs=100, patience=5)

ddr_model = DDR(X_train.shape[1], cutpoints)
train(mdn_model, train_dataset, val_dataset, epochs=100, patience=5)
```

Code: Distributional forecasts

```
glm_model.distributions(X_test[[1]])
```

```
Gamma(concentration: tensor([0.30]), rate: tensor([0.14]))
```

```
cann_model.distributions(X_test[[1]])
```

```
Gamma(concentration: tensor([0.29]), rate: tensor([0.14]))
```

```
mdn_model.distributions(X_test[[1]])
```

```
MixtureSameFamily(
  Categorical(probs: torch.Size([1, 5]), logits: torch.Size([1, 5])),
  Gamma(concentration: torch.Size([1, 5]), rate: torch.Size([1, 5])))
```

```
ddr_model.distributions(X_test[[1]])
```

```
Histogram(cutpoints: torch.Size([27]), prob_masses: torch.Size([1, 26]))
```

```
drn_model.distributions(X_test[[1]])
```

```
ExtendedHistogram(baseline: Gamma(concentration: tensor([0.30]), rate:
tensor([0.14])), cutpoints: torch.Size([27]), prob_masses: torch.Size([1,
26]))
```

Conclusion

- More than just mean predictions
- Checkout Eric's `drn` package on pypi (major update coming in 2-3 weeks)
- Suggestions and questions welcome, thanks for your attention!