ICEnets and Regularization

Deep Learning for Actuarial Modeling
36th International Summer School SAA
University of Lausanne

Ronald Richman, Salvatore Scognamiglio, Mario V. Wüthrich

11 September 2025

ICEnet - Producing Rational Results with Neural Networks

- Can neural networks be trained to produce actuarially sound results?
- The challenge: Deep learning models can learn “irrational" patterns that do not align with our expectations
- ICEnet (Richman & Wüthrich, 2024) solution: Constrain neural networks during training in the output space
- Key questions to discuss:
  - How can we enforce smoothness and monotonicity in neural network predictions?
  - Can we maintain or improve predictive accuracy with constraints?
- Bridge traditional actuarial techniques with modern ML

Regularization overview

We will cover **regularization concepts** for neural networks which closely connect to ICEnet A general formulation for the regularized empirical risk is:

$$\widehat{\theta} \in \arg\min_{\theta} \left(\sum_{i=1}^{n} \frac{v_i}{\varphi} L(y_i, \mu_\theta(x_i)) + \eta R(\theta_{-0}) \right),$$

where:

- Parameter vector: $\theta = (\theta_0, \theta_1, \ldots, \theta_r)^\top$ with intercept θ_0 (bias term)
- Regularise only $\theta_{-0} = (\theta_1, \ldots, \theta_r)^\top$; the intercept should not be penalised
- Scale-free η: drop any $1/n$ in the empirical loss so the strength of η does not diminish with n

- Examples: $R(\cdot) = \|\cdot\|_2^2$ (ridge Tikhonov and Arsenin (1977)), $\|\cdot\|_1$ (Lasso Tibshirani (1996)), $\|\cdot\|_0$ (best subset)
- See also Hastie, Tibshirani, and Wainwright (2015)

Roadmap of Lecture

- Introduce regularization concepts for general models (neural networks, GLMs, etc.)
- Explain basic ML interpretability techniques (PDPs, ICEs)
- Present the ICEnet architecture
- Show how to enforce constraints through loss functions
- Demonstrate results on French MTPL data
- Discuss extensions and improvements

Why We Need Regularization

- **The Goal**: Build models that generalize well to new, unseen data, not just the data they were trained on.
- **The Problem**: Models, especially complex or highly flexible ones, can easily **overfit**.
- i.e. These models may learn the random noise in the training data instead of the true underlying patterns.
- **Regularization** = set of methods designed to control model complexity.
- Many regularization methods prevent overfitting by penalizing large, unstable coefficients in some manner.

What is Regularization?

- Regularization is a way to formally manage the bias-variance tradeoff.
- **How it works**: We add a **penalty term** to the model's objective function, such as the negative log-likelihood (NLL).

$$\text{Minimize:} \quad \text{NLL}(\beta) + \text{Penalty}(\beta)$$

- This penalty discourages models that don't conform with certain features that we desire.
- **The Bargain**: We deliberately introduce a small amount of **bias** into our coefficient estimates to achieve a much larger and more beneficial reduction in **variance**.

Bias vs. Variance ⇆

Under squared-error loss with $Y = f(X) + \varepsilon$ and $\mathbb{E}[\varepsilon \mid X] = 0$, the expected test MSE decomposes as (Bishop, 2006):

$$\mathbb{E}\big[(Y - \hat{f}(X))^2\big] = \underbrace{\mathrm{Bias}^2[\hat{f}(X)]}_{(\mathbb{E}_{\text{train}}\hat{f}(X) - f(X))^2} + \underbrace{\mathrm{Var}[\hat{f}(X)]}_{\mathrm{Var}_{\text{train}}(\hat{f}(X))} + \underbrace{\mathrm{Var}(\varepsilon \mid X)}_{\text{Irreducible error}},$$

- **Bias (underfitting)**: systematic error from an overly restrictive model or strong regularization.
- **Variance (overfitting)**: sensitivity to sampling noise; predictions vary a lot across training sets.
- **Tradeoff**: increasing flexibility typically lowers bias and raises variance; pick complexity to minimize expected test error.
- In ML, goal is to find the optimal balance that minimizes the *total error* on future data **BUT** in actuarial science, we cannot usually accept “biased" models!

L2 Regularization (Ridge): Shrinkage & Stability

- **Objective Function**: Ridge adds a penalty proportional to the **sum of the squared coefficients**, also known as the squared L2-norm:

$$\text{Minimize:} \quad \text{NLL}(\beta) + \lambda \sum_{j=1}^{p} \beta_j^2$$

- **The L2 Penalty**:
  - The $\sum \beta_j^2$ term penalizes large coefficient values.
  - The tuning parameter λ controls the strength of this penalty.
- **Primary Effect**: **Shrinkage**
  - All coefficients are shrunk towards zero but never reach it. This makes the model more stable.

Ridge Regression in Action

- For an Ordinary Least Squares (OLS) model, the Ridge solution has a closed form:

$$\hat{\beta}_{\text{Ridge}} = (X^T X + \lambda I)^{-1} X^T y$$

- **Solving Multicollinearity**: In standard OLS, if predictors are highly correlated, the $X^T X$ matrix is unstable and difficult to invert.
- The addition of the λI term adds a constant to the diagonal of the matrix, ensuring it is always invertible and stable. This leads to more plausible parameter estimates.

Ridge-Regularised Poisson Log-Link GLM

- Model: $\log \mu_\theta(x) = \theta_0 + \sum_{j=1}^{q} \theta_j x_j$ (do not penalise θ_0)

$$\widehat{\theta}^{\text{ridge}} = \underset{\theta \in \mathbb{R}^{q+1}}{\arg\min} \sum_{i=1}^{n} 2 v_i \left(\mu_\theta(x_i) - y_i - y_i \log \frac{\mu_\theta(x_i)}{y_i} \right) + \eta \sum_{j=1}^{q} \theta_j^2$$

L1 Regularization (Lasso): Sparsity & Feature Selection

- **Objective Function**: Lasso adds a penalty proportional to the **sum of the absolute values of the coefficients**, also known as the L1-norm:

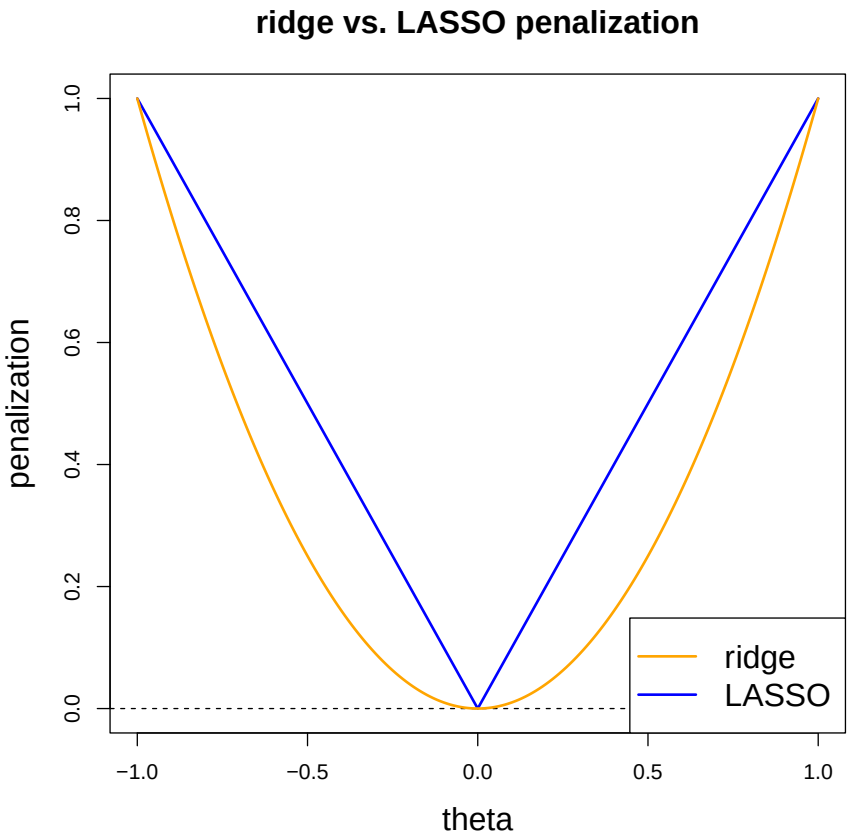
$$\text{Minimize:} \quad \text{NLL}(\beta) + \lambda \sum_{j=1}^{p} |\beta_j|$$

- **The L1 Penalty**:
  - The use of the absolute value $|\beta_j|$ is the key difference.
  - This penalty is non-differentiable at zero, which allows it to shrink some coefficients to exactly zero.
- **Primary Effect**: **Sparsity**
  - Lasso performs **automatic feature selection** by setting some coefficients exactly to zero, creating simpler, more interpretable models.

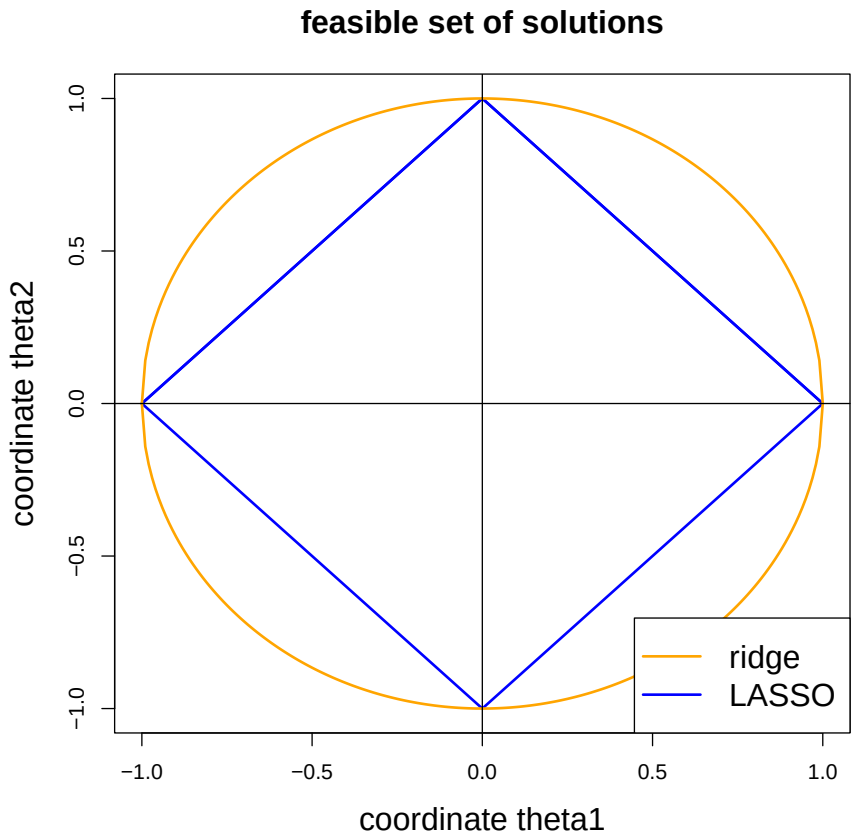
The Geometry of L1 vs. L2

The behavior of Ridge and Lasso can be understood by their geometric constraint regions.

Differentiability



Solution Space



The Fused Lasso: For Ordered Data

- **The Problem**: Standard methods ignore the natural ordering of predictors like age bands or bonus-malus levels.
- **Objective Function**: The Fused Lasso adds a second penalty for the differences between adjacent coefficients:

$$\text{Minimize:} \quad \text{NLL}(\beta) + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=2}^{p} |\beta_j - \beta_{j-1}|$$

- The **fusion penalty** ($\lambda_2 \sum |\beta_j - \beta_{j-1}|$) encourages adjacent coefficients to become equal, effectively "fusing" categories.
- This automates **tariff simplification** in a principled way.
- Encourages piecewise-constant effects over ordered categories; see Tibshirani, Saunders, Rosset, Zhu, and Knight (2005)

Best-Subset (L0) and Elastic Net

Best-subset (L0)

$$\min \ \text{NLL}(\beta) + \eta \sum_{j=1}^{r} \mathbf{1}\{\beta_j \neq 0\} \quad \text{(computationally hard)}$$

Elastic net (Zou & Hastie, 2005)

$$\min \ \text{NLL}(\beta) + \eta \left(\alpha \|\beta\|_1 + (1-\alpha)\|\beta\|_2^2 \right), \ \ \alpha \in [0,1]$$

Commentary

- L0 yields the sparsest model but is non-convex/combinatorial; small-p problems admit exact search, otherwise use heuristics; tune η via CV
- Elastic Net bridges L1/L2 (α=1 lasso, α=0 ridge); stabilises selection under collinearity and encourages grouping of correlated predictors Zou and Hastie (2005)

Group LASSO

Group LASSO (Yuan & Lin, 2006)

$$\min \sum_{i=1}^{n} \frac{v_i}{\varphi} L(y_i, \mu_\theta(x_i)) + \sum_{k=1}^{G} \eta_k \left\| \theta^{(k)} \right\|_2$$

Commentary

- Group LASSO selects/drops whole blocks (e.g., all dummies of a categorical); standardise within/between groups so penalties are comparable Yuan and Lin (2006)

An Actuarial Original - Whittaker-Henderson

- The principles of regularization have been part of actuarial science for over a century.
- **Whittaker-Henderson (WH) smoothing** for mortality graduation is a form of penalized least squares.
- **The WH Formula**: It minimizes a function that balances two goals:

$$\text{Minimize:} \quad \sum_x w_x (q'_x - q_x)^2 + \lambda \sum_x (\Delta^z q_x)^2$$

  - **Fidelity**: The first term, $\sum w_x (q'_x - q_x)^2$, measures the fit to the raw data q'_x.
  - **Smoothness**: The second term, $\lambda \sum (\Delta^z q_x)^2$, is a **penalty** that discourages roughness in the graduated rates q_x.

From Regularization to ICEnet: The Next Step

- **What We've Seen**:
  - Classic regularization methods control the **coefficients** of a GLM.
  - They add penalties to an objective function to enforce desired properties like **stability** (Ridge), **simplicity** (Lasso), and **smoothness** (Fused Lasso).
- **The New Challenge**:
  - In a complex "black box" model like a neural network, we can't directly control the coefficients in the same way.
- **The Question**:
  - How can we apply this same principle - penalizing undesirable behavior in a loss function - to control the **outputs** of the model to ensure they are rational and smooth?

Families of Neural Network Regularisation

This section explores the primary methods used to control complexity and prevent overfitting directly within neural network architectures. We can group these techniques into three main families:

- **Explicit Regularisation**: Directly modifying the loss function to add a penalty for complexity.
- **Stochastic Regularisation**: Injecting randomness into the training process to build more robust models.
- **Implicit Regularisation**: Regularising effects that arise naturally from the choice of model architecture and optimization algorithm.

Explicit Regularisation: L2 or L1

The most direct way to regularise a network is as above: by adding a penalty term to the loss function. L2 regularisation, the most common form, directly penalizes the sum of the squared weights.

- **The Intuition**: A model with smaller weights is simpler and less sensitive to minor changes in input features, making it more stable.
- **Objective Function**: The standard data loss, $L_{data}(w)$, is augmented with the squared L2-norm of the weights:

$$L_{total}(w) = L_{data}(w) + \frac{\lambda}{2} \sum_i w_i^2$$

Here, λ is the regularisation rate that controls the strength of the penalty.

Can also use L1 regularisation, which penalizes the sum of the absolute values of the weights!

Weight Decay vs L2 (summary)

- **L2-regularised objective**: $L_{total}(w) = L_{data}(w) + \frac{\lambda}{2} \sum_i w_i^2$
- **Weight decay SGD update**:
  $w \leftarrow w - \eta(\nabla L_{data}(w) + \lambda w) = (1 - \eta\lambda)w - \eta \nabla L_{data}(w)$
  Equivalence: For plain SGD, L2 penalty and classical weight decay are equivalent
- **Decoupled weight decay (AdamW)** Loshchilov and Hutter (2019):
  - Remove the regularization term from the gradient update
  - Needed for adaptive optimizers; differs from L2 penalty because the penalty gradient would be rescaled per-parameter
- **Bayesian view**: L2 penalty corresponds to a zero-mean Gaussian prior on weights

Stochastic Regularisation: Dropout

Stochastic methods regularise by injecting randomness into the training process, forcing the network to learn more robust features. **Dropout** is the most prominent example.

- **The Mechanism**: During each training step, a random fraction of neurons are temporarily "dropped" (their outputs are set to zero).
- **The Intuition**: This prevents neurons from developing complex co-adaptations, where one neuron relies on another to function correctly. Each neuron must learn features that are more generally useful and robust on their own.
- **Ensemble Interpretation**: Dropout is an efficient way of training a massive ensemble of different "thinned" neural networks. At test time, using the full network approximates averaging the predictions of all these sub-networks, a powerful technique for reducing overfitting.
- Acts only during training; see Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov (2014); Wager, Wang, and Liang (2013)

Dropout: Mathematical Formulation

Training-time

- For each hidden activation h_i, sample mask $m_i \sim \text{Bernoulli}(p_{\text{keep}})$ independently
- Scale to preserve expectation: $\tilde{h} = \dfrac{m \odot h}{p_{\text{keep}}}$

Moments

$$\mathbb{E}[\tilde{h}_i] = h_i, \qquad \text{Var}(\tilde{h}_i) = \frac{1 - p_{\text{keep}}}{p_{\text{keep}}} h_i^2$$

For a linear pre-activation $z = w^\top \tilde{h} + b$:

$$\mathbb{E}[z] = w^\top h + b, \qquad \text{Var}(z) = \sum_i w_i^2 \, \text{Var}(\tilde{h}_i)$$

Test-time

- Use full activations h without masks (no scaling needed)

Stochastic Regularisation: Batch Normalization

While primarily designed to stabilize and accelerate training, **Batch Normalization (BatchNorm)** also provides a subtle regularising effect.

- **Primary Goal**: BatchNorm addresses "internal covariate shift" by normalizing the inputs to each layer to have zero mean and unit variance for each mini-batch.
- **Regularising Effect**: The effect comes from the mean (μ_B) and variance (σ_B^2) being calculated on a random mini-batch, not the entire dataset.
  - The same data point will be normalized slightly differently depending on the other points in its mini-batch.
  - This injects a small amount of noise into the layer's activations at each step, making it harder for the model to memorize the training data.

Implicit Regularisation: The Optimizer and Architecture

Implicit regularisation arises naturally from the training process itself, without any explicit penalty or added noise.

- **From the Optimizer (SGD)**: The choice of optimizer has a profound impact.
  - Loss landscapes can have sharp minima (brittle solutions that don't generalize) and flat minima (robust solutions that do).
  - The inherent noise in **Stochastic Gradient Descent (SGD)** biases the optimizer towards finding the wider, flatter minima that are associated with better generalization.
- **From the Architecture**: The structure of a deep network can induce a preference for simple solutions.
  - Even when over-parameterized, networks trained with gradient descent show a bias towards finding simple functions.
  - For example, training deep linear networks implicitly biases the solution towards a **low-rank matrix**, corresponding to a simpler, more structured relationship.

Implicit Regularisation: One-slide

Optimizer (SGD)

- Flat minima correlate with better generalisation (Hochreiter & Schmidhuber, 1997; Li, Xu, Taylor, Studer, & Goldstein, 2018).
- Small-batch SGD tends to find flatter minima than large-batch (Keskar, Mudigere, Nocedal, Smelyanskiy, & Tang, 2017; aw Jastrzębski et al., 2018).
- SGD noise can be viewed as approximate Bayesian inference (Mandt, Hoffman, & Blei, 2017).
- Tuning the noise scale improves test performance (Smith, Elsen, & De, 2020).

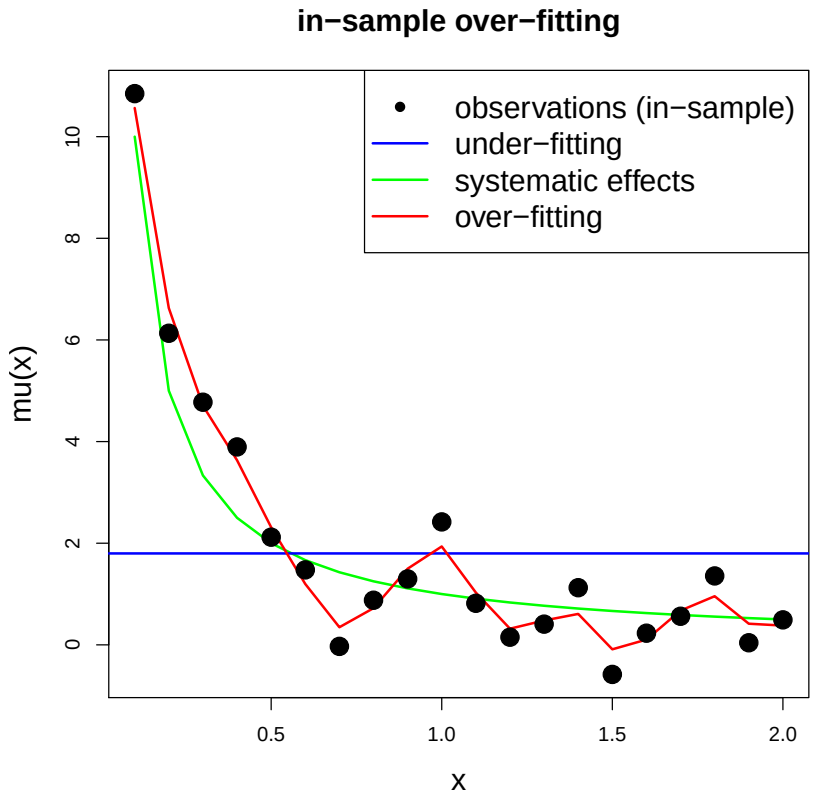
Architecture

- Parameter-to-function map is biased toward simple functions (Valle-Pérez, Camargo, & Louis, 2019).
- Spectral bias: low-frequency components are learned first (Rahaman et al., 2019).
- Deep linear/factorised models select low-rank, near minimum nuclear-norm solutions under gradient descent (Gunasekar, Woodworth, Bhojanapalli, Neyshabur, & Srebro, 2017; Arora, Cohen, Hu, & Luo, 2019).

Early Stopping

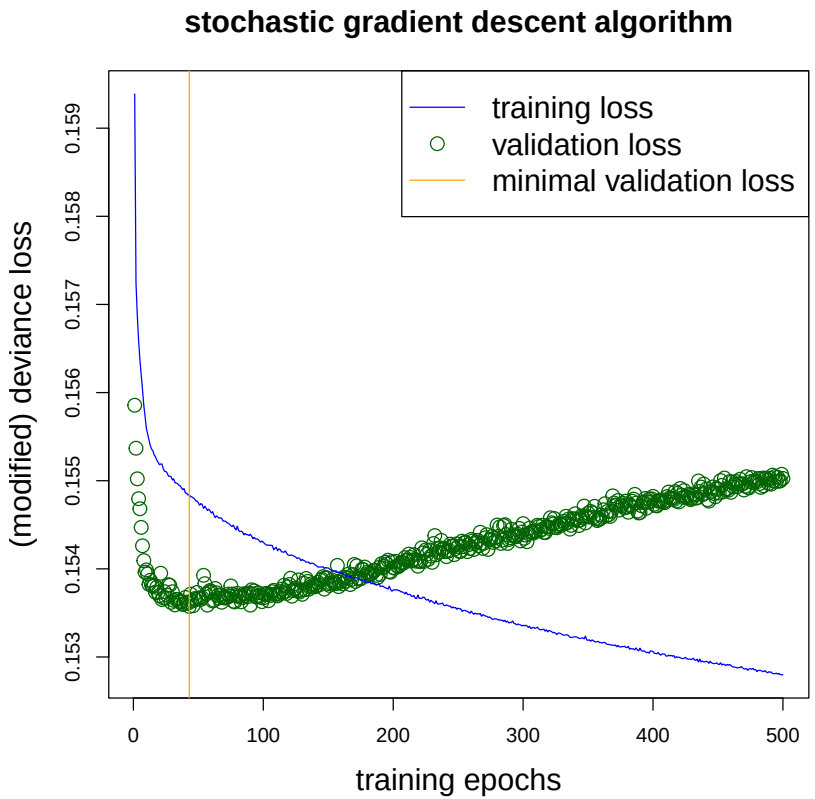
- Early stopping is a simple and effective way to prevent overfitting.
- Stop the training process when the performance on a validation set stops improving.
- i.e. we do not allow model fitting to converge!
- Early fitting iterations learn **systematic effects** common to many instances; later iterations chase **noise**
- Stop when validation loss stops improving to avoid fitting noise

Overfitting Illustration



Validation Split and Stopping Rule

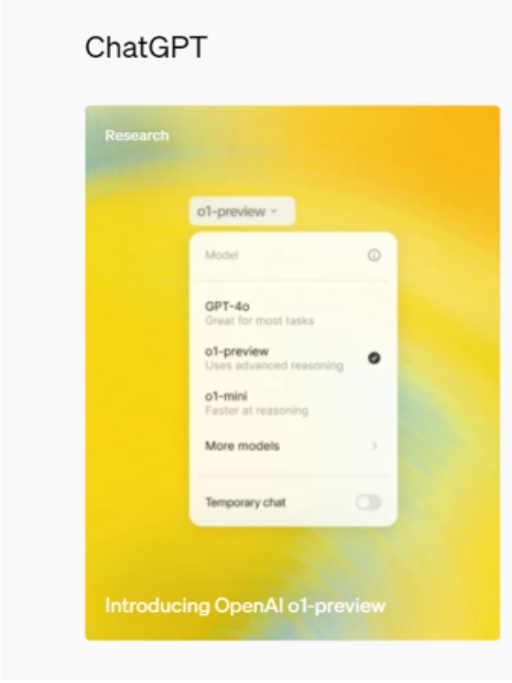
- Split learning data into training $\mathcal{U}$ and validation $\mathcal{V}$
- Train on $\mathcal{U}$, monitor validation loss $L(\theta^{[t]}; \mathcal{V})$
- Use a **callback**: keep the parameters with minimal validation loss



Presentation Roadmap

- We will explain the ICEnet approach to constraining neural networks
- Explain the ICEnet structure =>
  - Use ML interpretability techniques during model training...
  - ...to ensure models are trained to capture relationships appropriately
- Test models on French Motor Third Party Liability (MTPL) data

Wider field of ensuring "rational results"



Deep Reinforcement Learning from Human Preferences

Paul F Christiano
OpenAI
paul@openai.com

Jan Leike
DeepMind
leike@google.com

Tom B Brown
nottombrown@gmail.com

Miljan Martic
DeepMind
miljanm@google.com

Shane Legg
DeepMind
legg@google.com

Dario Amodei
OpenAI
damodei@openai.com

Abstract

For sophisticated reinforcement learning (RL) systems to interact usefully with real-world environments, we need to communicate complex goals to these systems. In this work, we explore goals defined in terms of (non-expert) human preferences between pairs of trajectory segments. We show that this approach can effectively solve complex RL tasks without access to the reward function, including Atari games and simulated robot locomotion, while providing feedback on less than 1% of our agent's interactions with the environment. This reduces the cost of human oversight far enough that it can be practically applied to state-of-the-art RL systems. To demonstrate the flexibility of our approach, we show that we can successfully train complex novel behaviors with about an hour of human time. These behaviors and environments are considerably more complex than any which have been previously learned from human feedback.

Training language models to follow instructions with human feedback

Long Ouyang* **Jeff Wu*** **Xu Jiang*** **Diogo Almeida*** **Carroll L. Wainwright***
Pamela Mishkin* **Chong Zhang** **Sandhini Agarwal** **Katarina Slama** **Alex Ray**
John Schulman **Jacob Hilton** **Fraser Kelton** **Luke Miller** **Maddie Simens**
Amanda Askell† **Peter Welinder** **Paul Christiano*†**
Jan Leike* **Ryan Lowe***
OpenAI

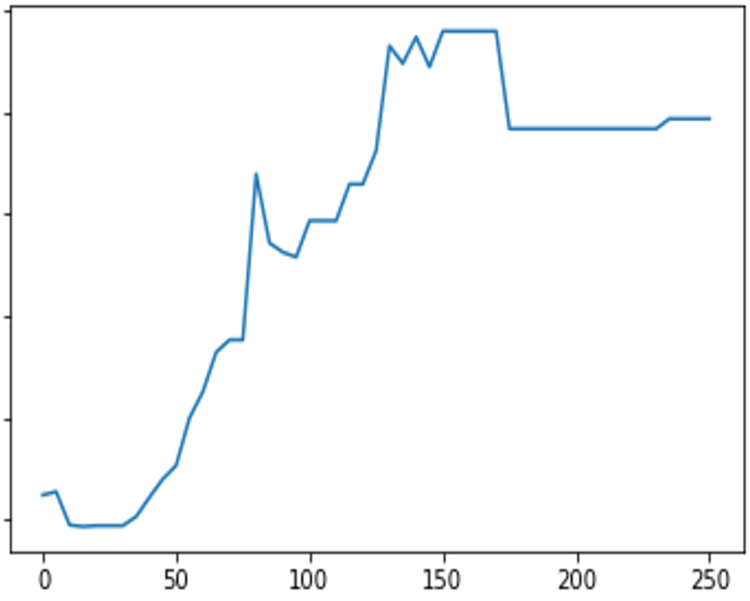
Abstract

Making language models bigger does not inherently make them better at following a user's intent. For example, large language models can generate outputs that are untruthful, toxic, or simply not helpful to the user. In other words, these models are not *aligned* with their users. In this paper, we show an avenue for aligning language models with user intent on a wide range of tasks by fine-tuning with human feedback. Starting with a set of labeler-written prompts and prompts submitted through the OpenAI API, we collect a dataset of labeler demonstrations of the desired model behavior, which we use to fine-tune GPT-3 using supervised learning. We then collect a dataset of rankings of model outputs, which we use to further fine-tune this supervised model using reinforcement learning from human feedback. We call the resulting models *InstructGPT*. In human evaluations on our prompt distribution, outputs from the 1.3B parameter InstructGPT model are preferred to outputs from the 175B GPT-3, despite having 100x fewer parameters. Moreover, InstructGPT models show improvements in truthfulness and reductions in toxic output generation while having minimal performance regressions on public NLP datasets. Even though InstructGPT still makes simple mistakes, our results show that fine-tuning with human feedback is a promising direction for aligning language models with human intent.

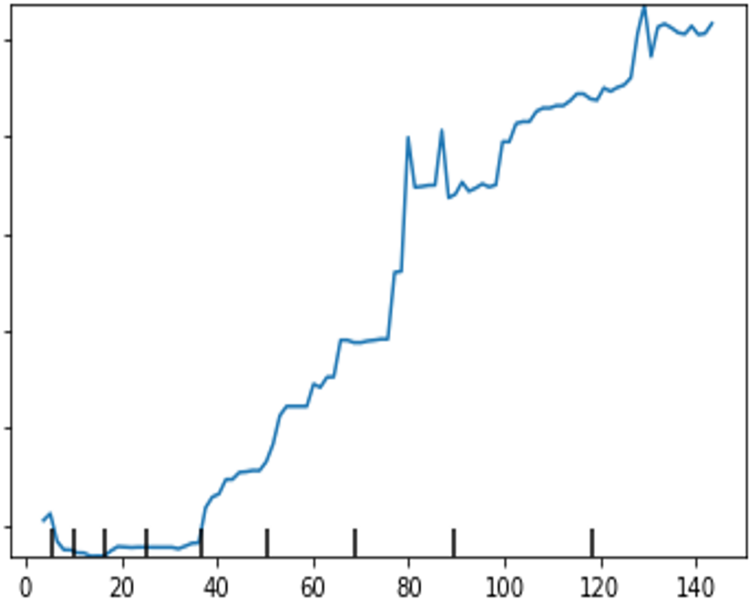
Pitfalls - 1

Climate Change Modelling with XGBoost

Sample P/H Sensitivity



Partial Dependency Plot



ML Interpretability

- Fitting an ML model means that model structure is opaque...
- ... so field of post-hoc model interpretability has developed
- = methods to determine how model has captured relationships between variables
- Techniques:
  - Partial Dependence Plots (PDPs)
  - Individual Conditional Expectations (ICEs)
  - SHapley Additive exPlanations (SHAP)
  - Local Interpretable Model-agnostic Explanations (LIME)
- Basic recipe:
  - Fit ML model
  - Modify the input data to model to isolate effect of single variable
  - Repeat for all combinations of data

ICEs and PDPs

- Main idea of ICEs is to test how model output varies. . .
- . . . as we vary one input variable
- Captures the marginal effect of changing a single variable's value
- ICE recipe:
  - Fit ML model
  - Modify value of single variable to isolate effect
  - Repeat for all records in the data
- PDPs = average of ICE over whole dataset
- Can be shown to be causal impact of changing a variable under certain assumptions (see Zhao and Hastie (2019))

ICE Plot Formula

$$\tilde{y}_n^{[j]} = [\Psi_W(\tilde{x}_n^{[j]}(1))v_n, ..., \Psi_W(\tilde{x}_n^{[j]}(K_j))v_n]$$

- $\tilde{y}_n^{[j]}$ = The Individual Conditional Expectation (ICE), which is a vector of predictions for a single policy n as feature j is varied across its possible values.
- Ψ_W = The trained neural network model, with all its learned weights and biases represented by W.
- $\tilde{x}_n^{[j]}(u)$ = The feature vector for the n-th policy, but with the value for feature j artificially set to a specific value u.
- v_n = The exposure for the specific n-th policy (e.g., duration in years).

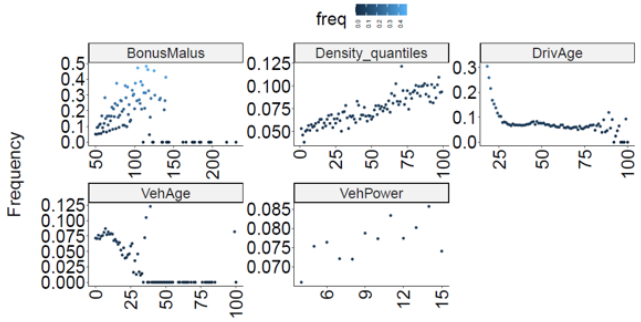
PDP Formula

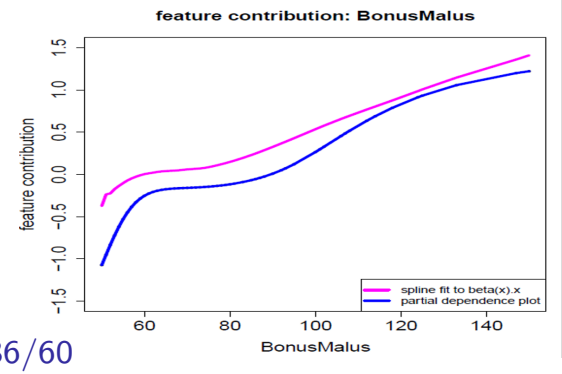
$$PDP^{[j]}(u) = \frac{1}{N} \sum_{n=1}^{N} \Psi_W(\tilde{x}_n^{[j]}(u)) v_n$$

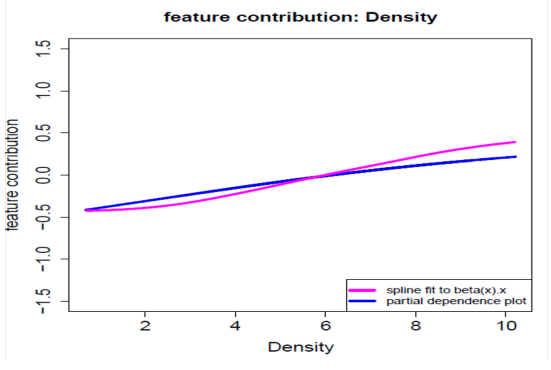
- $PDP^{[j]}(u)$ = The Partial Dependence Plot value for feature j at a specific value u.
- N = The total number of policies (or observations) in the dataset.
- Ψ_W = The trained neural network model, with all its learned weights and biases represented by W.
- $\tilde{x}_n^{[j]}(u)$ = The feature vector for the n-th policy, but with the value for feature j artificially set to u.
- v_n = The exposure for the n-th policy (e.g., duration in years).

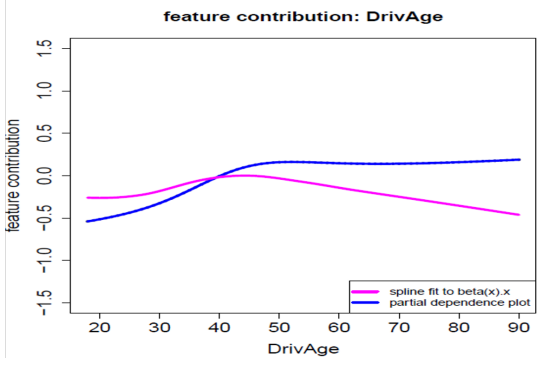
Example 1 - PDPs

- PDPs are different from plots of:
  - Empirical observations
  - Fitted model predictions
- PDPs isolate the effect of a single variable...
- ... whereas empirical/fitted plots are influenced by "correlation" between the input variables
- E.g. younger drivers have higher bonus-malus scores => high dependence between these variables removed by PDPs

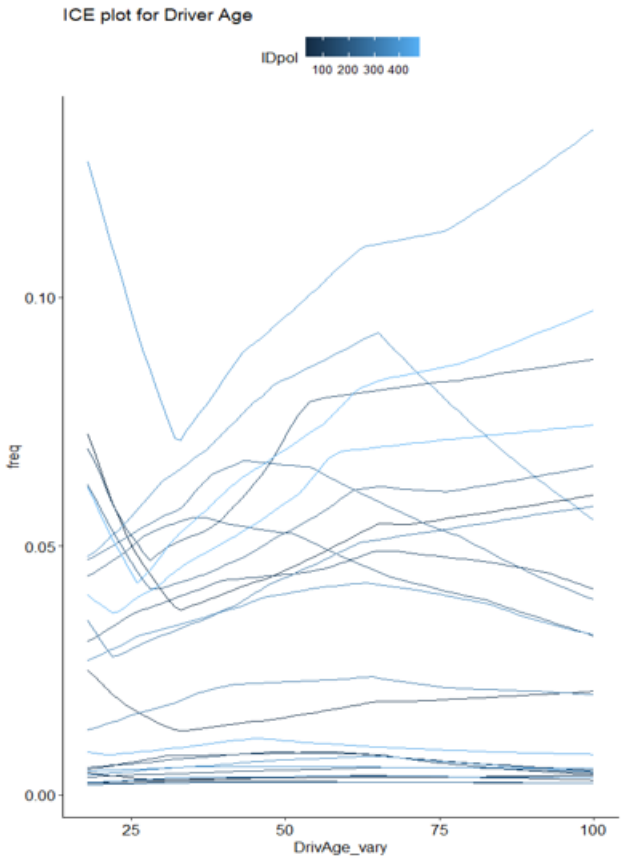


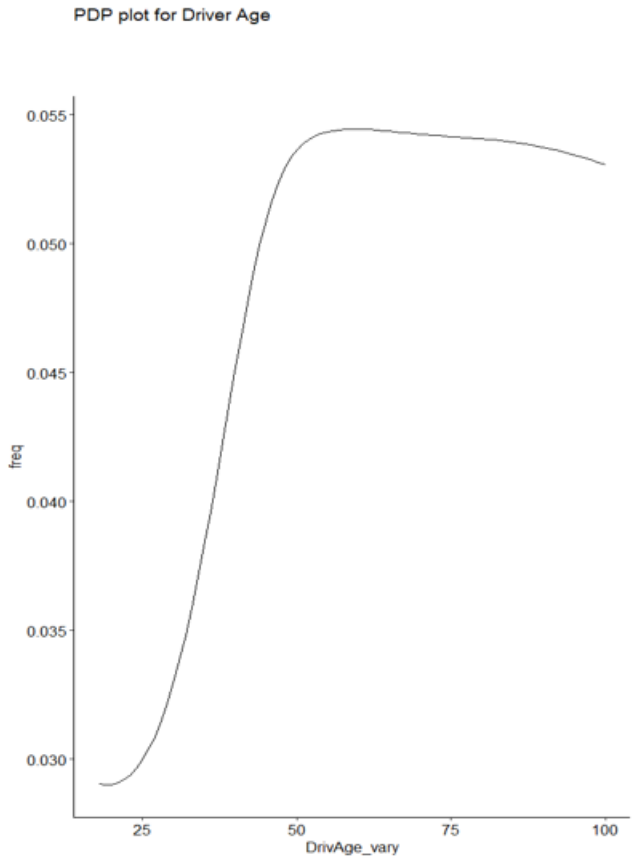


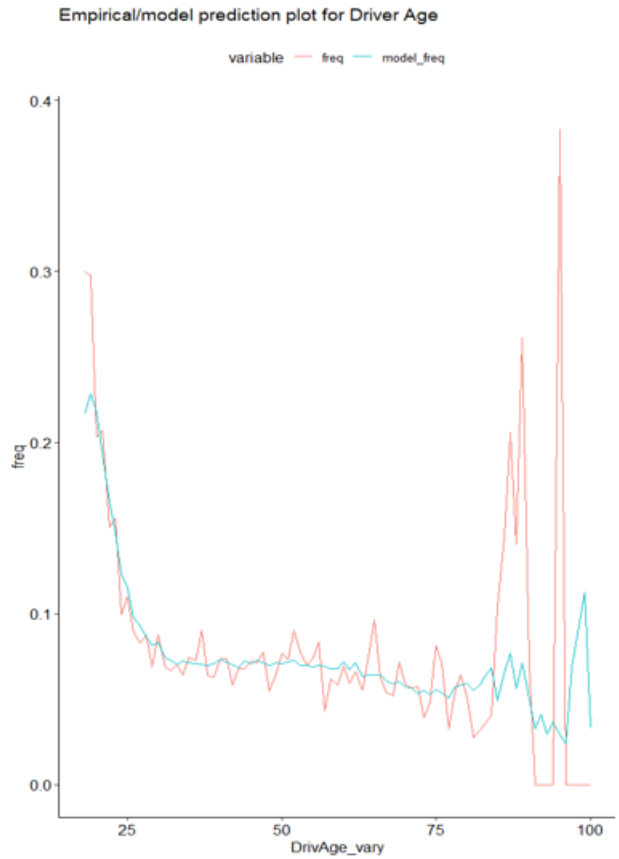




Example 2 – ICEs/PDPs/Empirical observations







Actuaries smooth PDPs = ICEs

- Actuarial pricing often uses GLMs which has an explicit linear model form (on scale of link function)

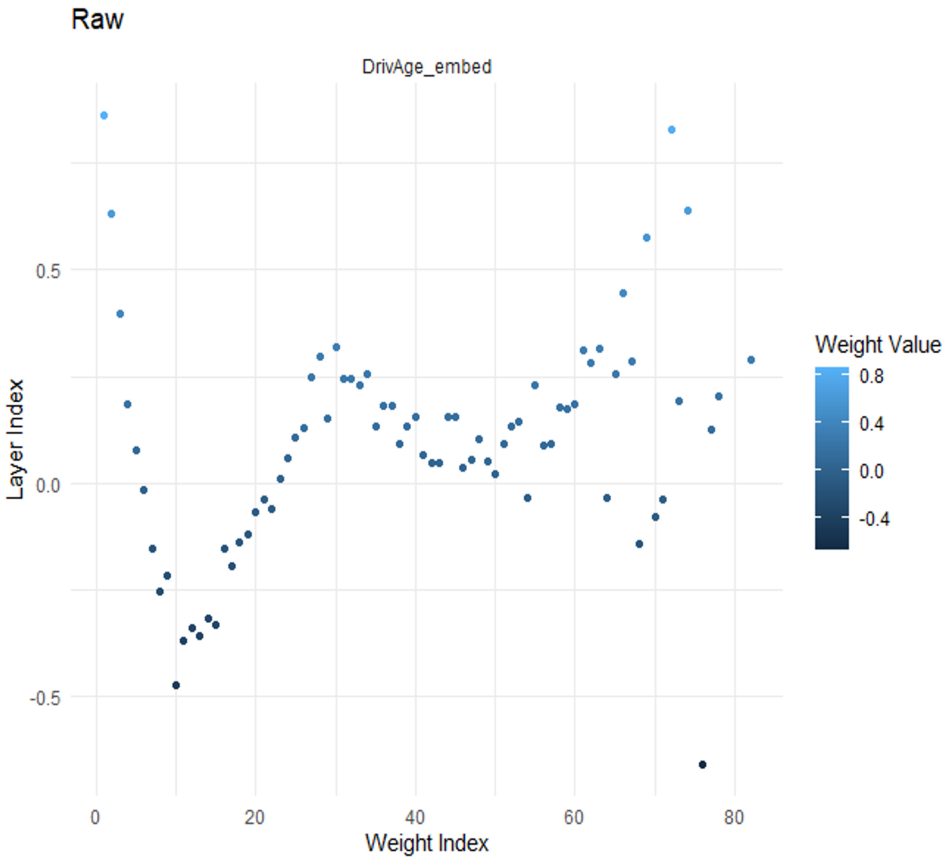
$$g^{-1}(y) = \beta_1.x_1 + \cdots + \beta_q.x_q$$

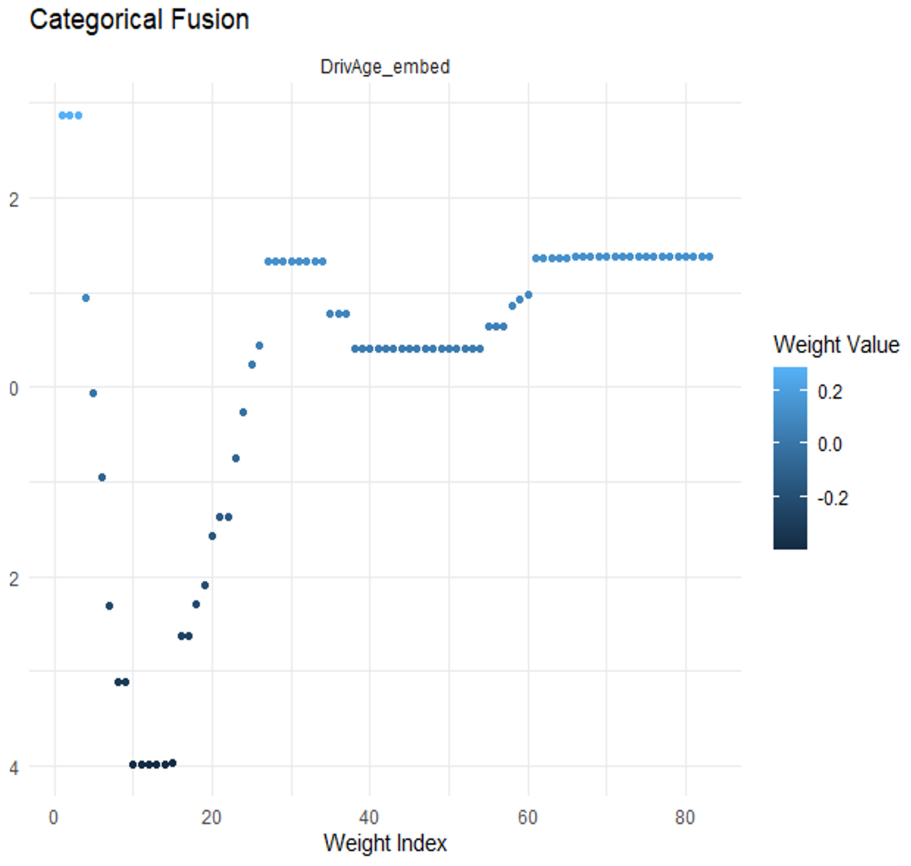
- Suppose x_1 is categorical (or numerical and was binned) then we can represent the GLM as:

$$g^{-1}(y) = \begin{cases} \beta_{1,1} + \cdots + \beta_q.x_q & \text{when } x_1 = x_{1,1} \\ \beta_{1,2} + \cdots + \beta_q.x_q & \text{when } x_1 = x_{1,2} \\ \cdots & \\ \beta_{1,q_1} + \cdots + \beta_q.x_q & \text{when } x_1 = x_{1,q_1} \end{cases}$$

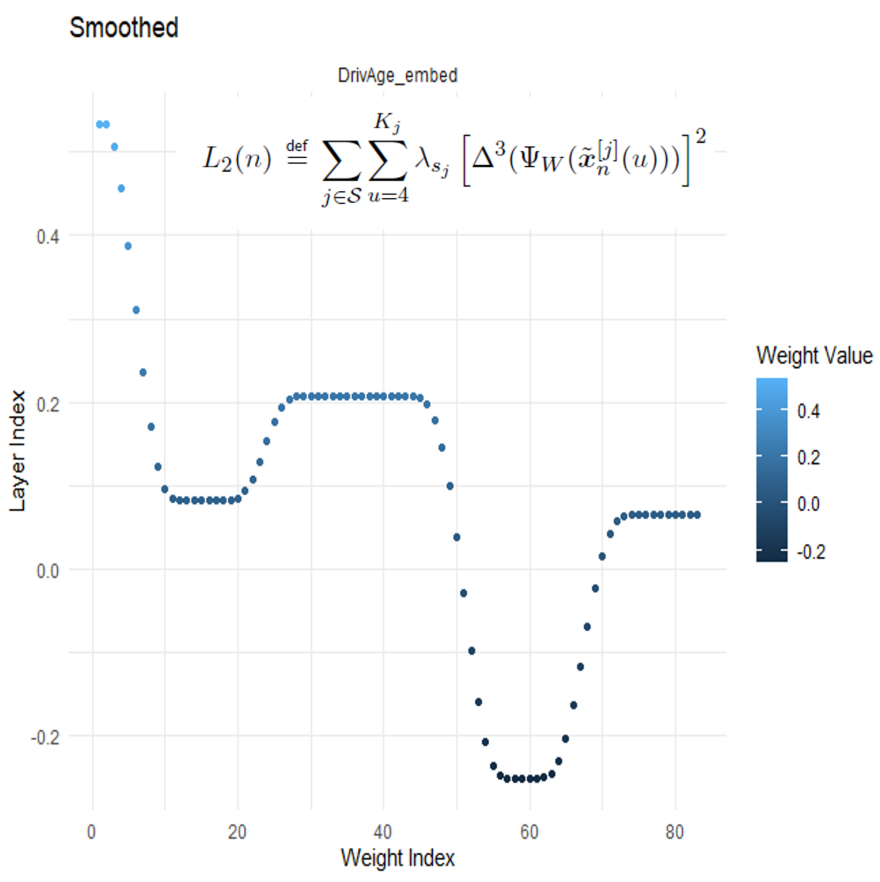
- In a GLM, as we vary $x_{1,1} \to x_{1,2}$ our GLM value changes by $\beta_{1,2} - \beta_{1,1}$
- PDPs and ICEs determined by GLM coefficients
- As we smooth or group GLM coefficients, we are smoothing the PDPs and the ICEs
- SHAP values can be derived directly from GLM + fitted parameters

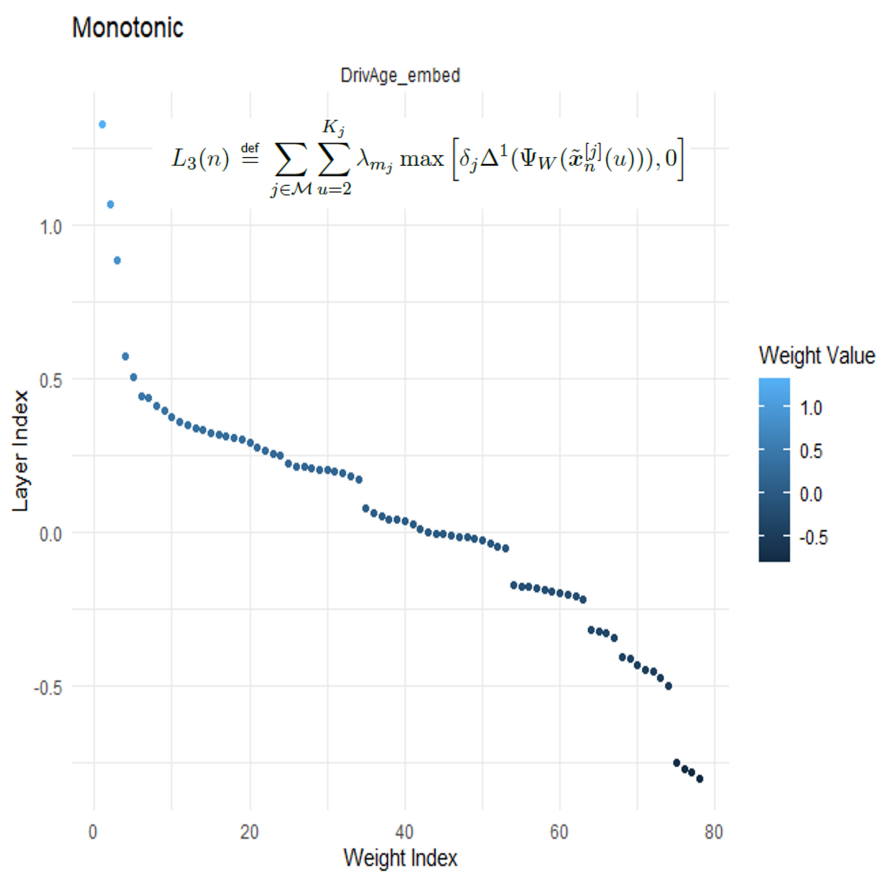
Grouping GLM coefficients





Grouping GLM coefficients

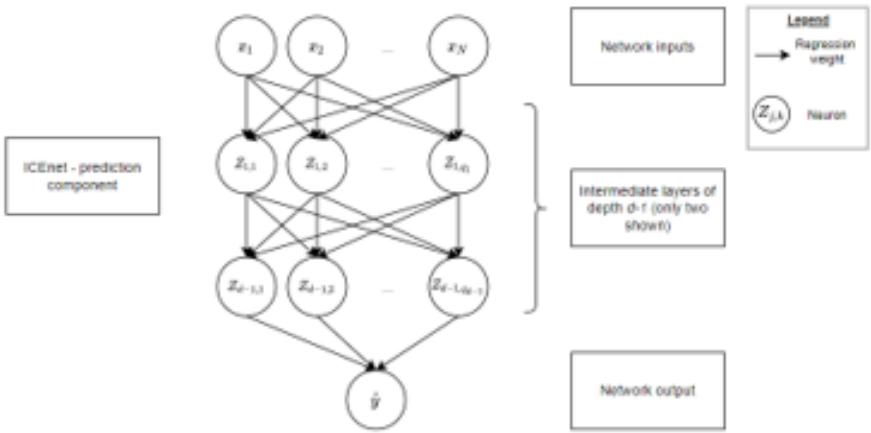




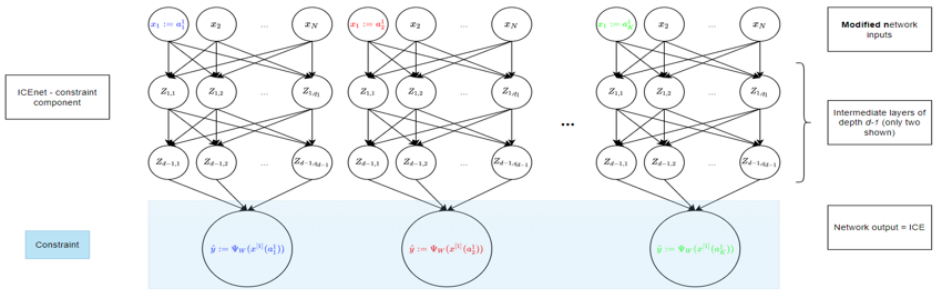
ICEnet - 1

- Neural network with two main components:
  - Prediction network
  - Constraint network

Prediction network = vanilla neural network



Constraint network = creates predictions using pseudo-data



- Parameters shared with constraint network => constraints influence predictions

ICEnet - 2

- For constraint network, we augment the data input to network...
- ... using exactly the same inputs we would use to derive an ICE = pseudo-data

$$\tilde{x}_n \mapsto \tilde{y}_n^{[s_1]}, ..., \tilde{y}_n^{[s_S]}, \tilde{y}_n^{[m_1]}, ..., \tilde{y}_n^{[m_M]}$$

  - $\tilde{x}_n$ = The input to the network's constraint component, representing the pseudo-data for a single policy n.
  - $\tilde{y}_n^{[s_1]}, ..., \tilde{y}_n^{[s_S]}$ = The set of output ICE (Individual Conditional Expectation) vectors for all features requiring a smoothness constraint (from the set $\mathcal{S}$).
  - $\tilde{y}_n^{[m_1]}, ..., \tilde{y}_n^{[m_M]}$ = The set of output ICE vectors for all features requiring a monotonicity constraint (from the set $\mathcal{M}$).

ICEnet Compound Loss Function

- Train network using compound loss function:
  - Prediction loss – ensure model makes good predictions (Whittaker-Henderson smoothing)
    - E.g. Poisson deviance for claims frequency
  - Smoothing loss – ensure model makes smooth predictions on the pseudo-data
  - Monotonicity loss – ensure model enforces monotonicity on the pseudo-data (Monotonicity loss – Penalize deviations)

ICEnet Compound Loss Function - Prediction Loss

Poisson Deviance Function:

$$L^D(y_n, \hat{y}_n) = 2\left[y_n \log\left(\frac{y_n}{\hat{y}_n}\right) - (y_n - \hat{y}_n)\right]$$

- $L^D(y_n, \hat{y}_n)$ = The Poisson deviance loss for a single observation n, which measures the goodness-of-fit of the model for that observation.
- y_n = The observed value (e.g., the actual number of claims) for the n-th policy.
- $\hat{y}_n$ = The model's predicted value (e.g., the expected number of claims) for the n-th policy.
- $\log(\cdot)$ = The natural logarithm function.
- Note: The term $y_n \log(y_n)$, which is part of the expanded formula, is taken to be zero when $y_n = 0$.

ICEnet Compound Loss Function - Smoothing Loss

Smoothing Loss Component:

$$L_2(n) = \sum_{j \in S} \sum_{u=4}^{K_j} \lambda_{s_j} [\Delta^3(\Psi_W(\tilde{x}_n^{[j]}(u)))]^2$$

- $L_2(n)$ = The smoothing loss component for a single policy n.
- S = The set of all features that require a smoothness constraint.
- K_j = The number of distinct values for feature j over which the constraint is evaluated.
- λ_{s_j} = The penalty parameter that controls the strength of the smoothing constraint for feature j.
- $[\Delta^3(\cdot)]^2$ = The squared third-order difference of the model's predictions. This term penalizes roughness; a smaller value implies a smoother, more linear relationship.

ICEnet Compound Loss Function - Monotonicity Loss

Monotonicity Loss Component:

$$L_3(n) = \sum_{j \in M} \sum_{u=2}^{K_j} \lambda_{m_j} \max[\delta_j \Delta^1(\Psi_W(\tilde{x}_n^{[j]}(u))), 0]$$

- $L_3(n)$ = The monotonicity loss component for a single policy n.
- M = The set of all features that require a monotonicity constraint.
- λ_{m_j} = The penalty parameter that controls the strength of the monotonicity constraint for feature j.
- $\max[\cdot, 0]$ = A function that ensures a penalty is only applied if the trend violates the desired monotonicity (i.e., the value inside is positive).
- δ_j = A direction parameter set to -1 to enforce a monotonic increase or $+1$ to enforce a monotonic decrease for feature j.

ICEnet Compound Loss Function (iv) Putting it All Together

Total ICEnet Loss:

$$L(n) = L^D(y_n, \hat{y}_n) + L_2(n) + L_3(n)$$

- $L(n)$ = The total compound loss for a single policy n, which the model aims to minimize during training.
- $L^D(y_n, \hat{y}_n)$ = The deviance loss (e.g., Poisson deviance), which measures the predictive accuracy of the model on the actual data.
- $L_2(n)$ = The smoothing loss, which penalizes non-smooth model outputs.
- $L_3(n)$ = The monotonicity loss, which penalizes model outputs that violate a desired monotonic trend.

ICEnet results - 1

- Fit ICEnet to French MTPL data
- Split data in 90/10 ratio for train/testing sets
- Used small neural network:
  - 3 layers – 32/16/8
  - Embeddings for categorical covariates
- Ensemble results = network aggregation (nagging)
- Value of smoothing/monotonicity constraints chosen to produce acceptably smooth/monotonic ICEs
- About 3 minutes to fit normal NN/12 minutes to fit ICEnet on GPU
- Small loss of predictive power in exchange for constraints (0.2385 vs 0.2383)

Set	N	Exposure	Claims	Frequency
learn	610,206	322,392	23,737	0.0736
test	67,801	35,967	2,644	0.0735

Covariate	Smoothing Constraint	Monotonicity Constraint	Direction
Driver Age	10	0	−1
Vehicle Age	1	0	−1
Bonus Malus	1	100	−1
Density	1	100	−1
Vehicle Power	1	100	−1

Description	Learn		Test	
FCN	0.2381	(0.000211)	0.2387	(0.000351)
FCN (nagging)	0.2376		0.2383	
ICEnet	0.2386	(0.000180)	0.2388	(0.000236)
ICEnet (nagging)	0.2384		0.2385	

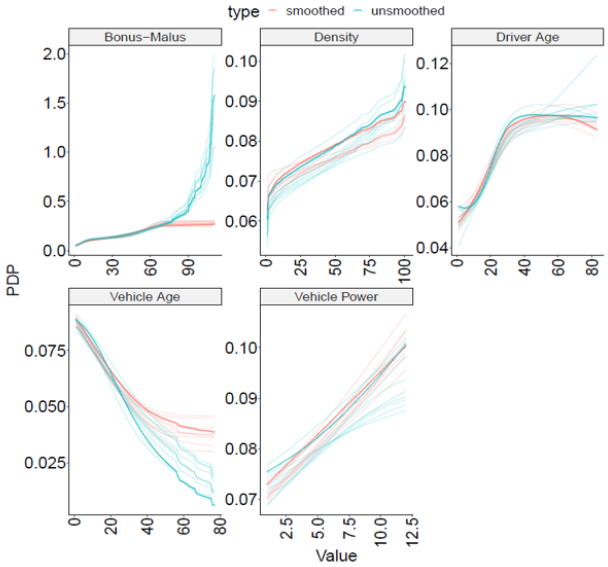
ICEnet results - 2

- Can we decompose the difference in predictive power between:
  - Smoothing constraint vs Monotonicity constraint?
- Fit 3 versions of the ICEnet (baseline performance = 0.2383)
- Smoothing + Monotonicity (worse predictive performance)/Only Smoothing (worse predictive performance)/Only Monotonicity (better predictive performance)

Description	Constraints	Learn		Test	
ICEnet	Smoothing + Monotonicity	0.2386	(0.000180)	0.2388	(0.000236)
ICEnet	Smoothing	0.2385	(0.000423)	0.2388	(0.000385)
ICEnet	Montonicity	0.2384	(0.000214)	0.2385	(0.000140)
ICEnet (nagging)	Smoothing + Monotonicity	0.2384		0.2385	
ICEnet (nagging)	Smoothing	0.2382		0.2385	
ICEnet (nagging)	Monotonicity	0.2381		0.2382	

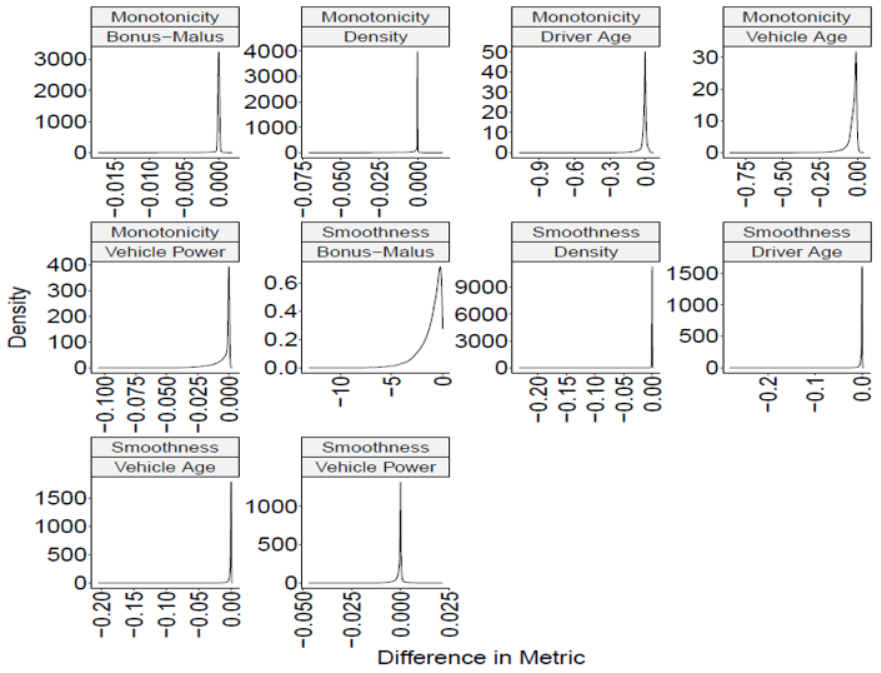
ICEnet PDPs

- PDPs from unconstrained NN models exhibit several undesirable characteristics:
  - Bonus-malus level/density/vehicle age covariates - significant roughness = unacceptable changes in rates
  - In some PDPs, monotonicity has not been maintained
  - PDPs for the driver age and vehicle power exhibit different shapes over the different runs = learned different relationships between the covariates
- PDPs from ICEnets significantly smoother and monotonically increasing in all cases with constraint
- Relationships learned by ICEnets quite different for bonus-malus level, vehicle age and driver age

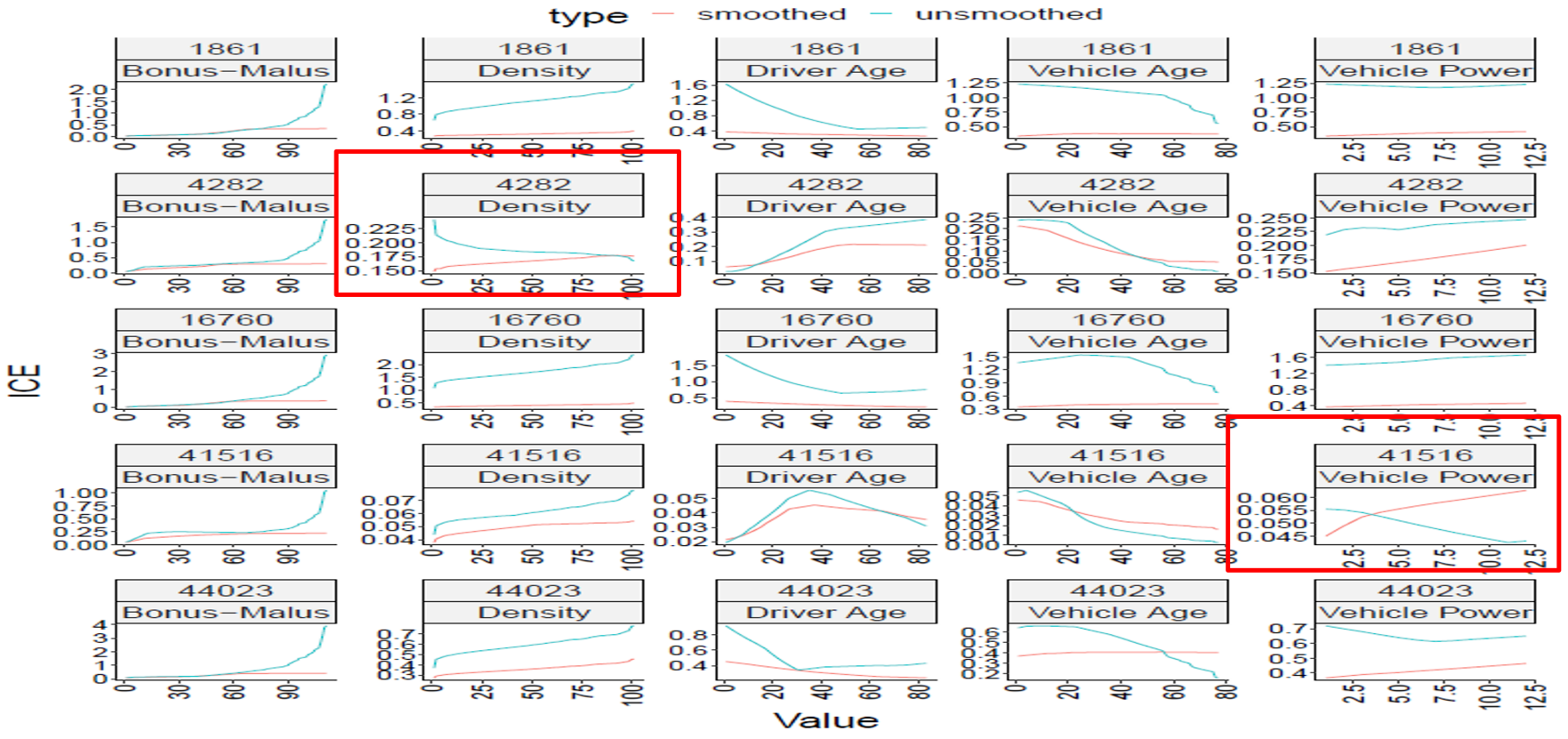


ICEnet – did it work?

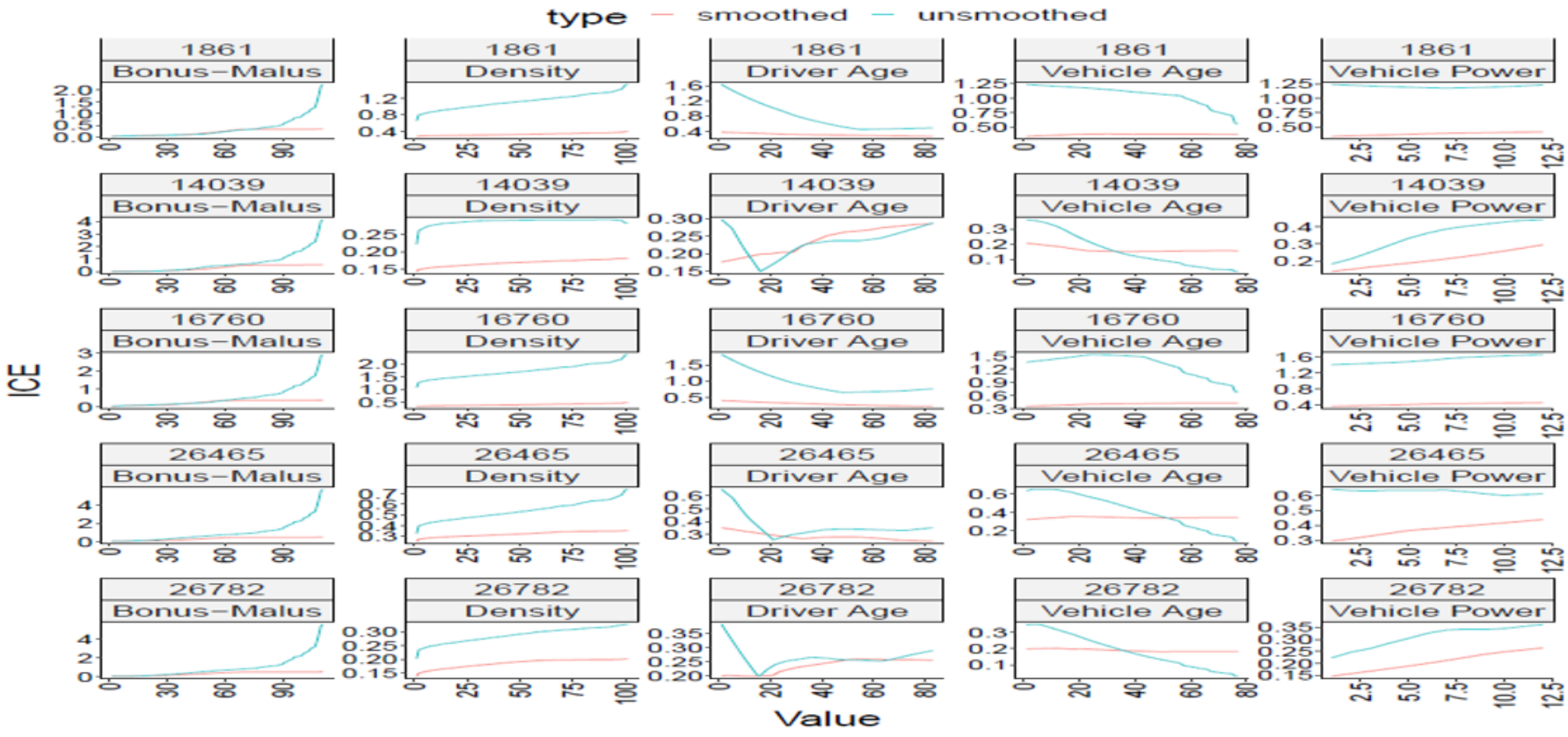
- Penalty on constrained ICE – Penalty on unconstrained ICE
- Densities have long left tail => ICE outputs significantly more monotonic and smooth
- Densities peak around zero => adding constraints has generally not significantly "damaged" outputs from FCN
- Densities of monotonicity scores for bonus-malus level and density variables = short right tail => original FCN model produces some outputs that are already monotonic and these are altered somewhat by ICEnet



ICEnet – ICEs - monotonicity results

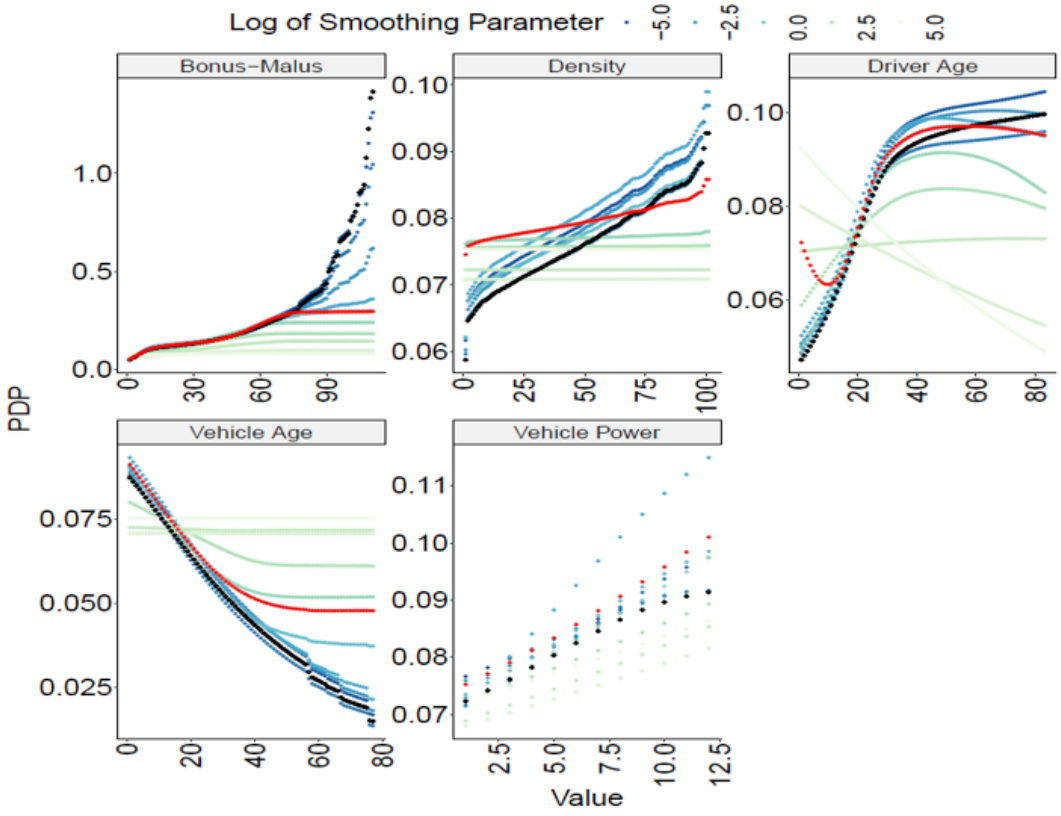


ICEnet – ICEs - smoothness results



Varying the penalties

- Fit an ICEnet while varying the value of the constraints from very small to very large
- PDPs shown for different values of the constraints; Unconstrained = black
- Best fit selected based on validation set and shown in red

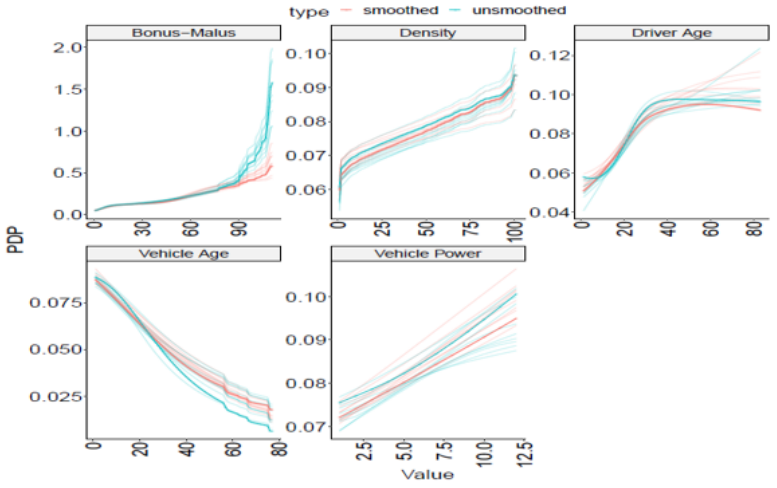


Regularization Parameter ($\log_{10}$)	Learn	Validation	Test
meta-model	0.23835	0.23232	0.23829
-5	0.23838	0.23276	0.23856
-4	0.23833	0.23245	0.23847
-3	0.23791	0.23243	0.23837
-2	0.23831	0.23263	0.23855
-1	0.23816	0.23241	0.23856
0	**0.23785**	**0.23236**	**0.23840**
1	0.23924	0.23332	0.23947
2	0.24093	0.23484	0.24099
3	0.24341	0.23684	0.24375
4	0.24659	0.23965	0.24744
5	0.24903	0.24212	0.25018

Local ICEnet

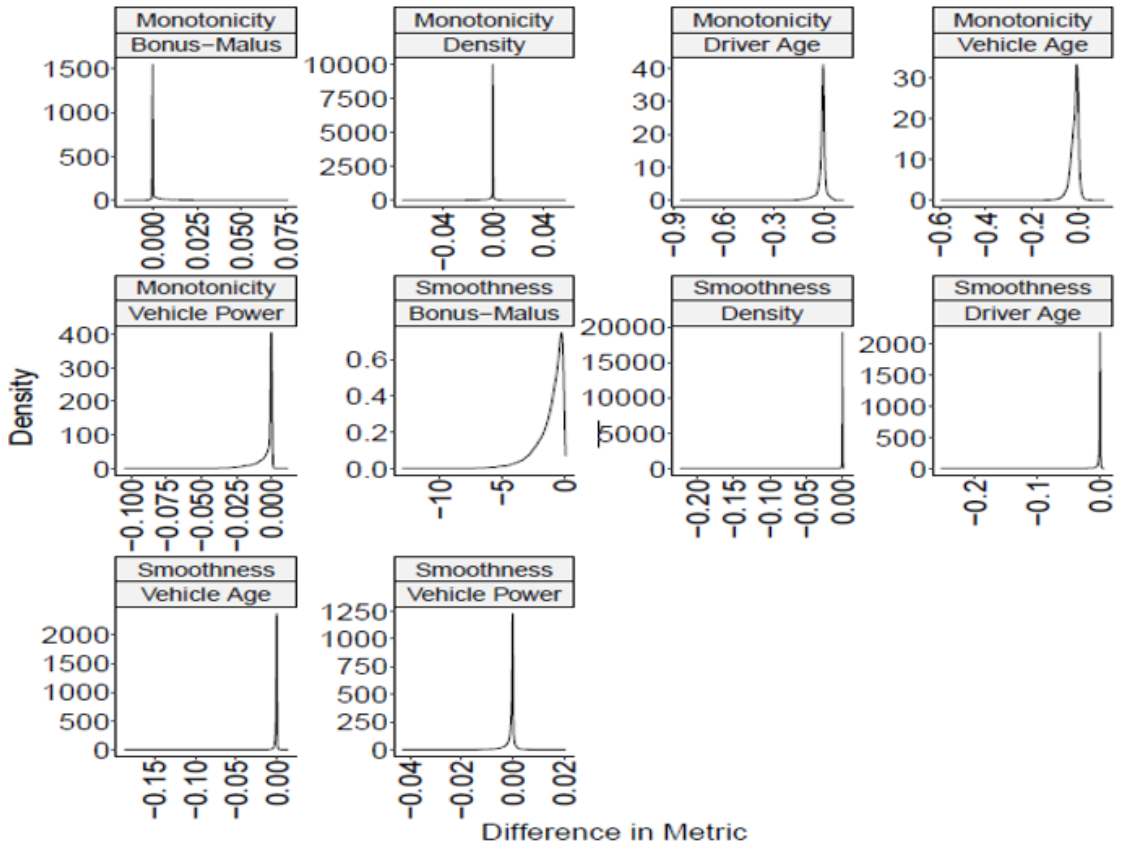
- ICEnet penalties act over whole range of the dataset $=>$
  - Influenced by values of variables with little exposure
  - Computationally heavy
- Local ICEnet – apply the same principle as ICEnet but only on a small "window" around each observation
- Parameter ω is chosen to synthesize $\frac{\omega-1}{2}$ data points on either side of actual data point
- 12 minutes to fit on CPU without GPU
- Local ICEnet outperforms NN on average and for nagging predictor
- $=>$ Local ICEnet allows constraints to be enforced with smaller loss of predictive

Description	Learn		Test	
FCN	0.2381	(0.000211)	0.2387	(0.000351)
FCN (nagging)	0.2376		0.2383	
ICEnet	0.2385	(0.000301)	0.2386	(0.000215)
ICEnet (nagging)	0.2381		0.2383	

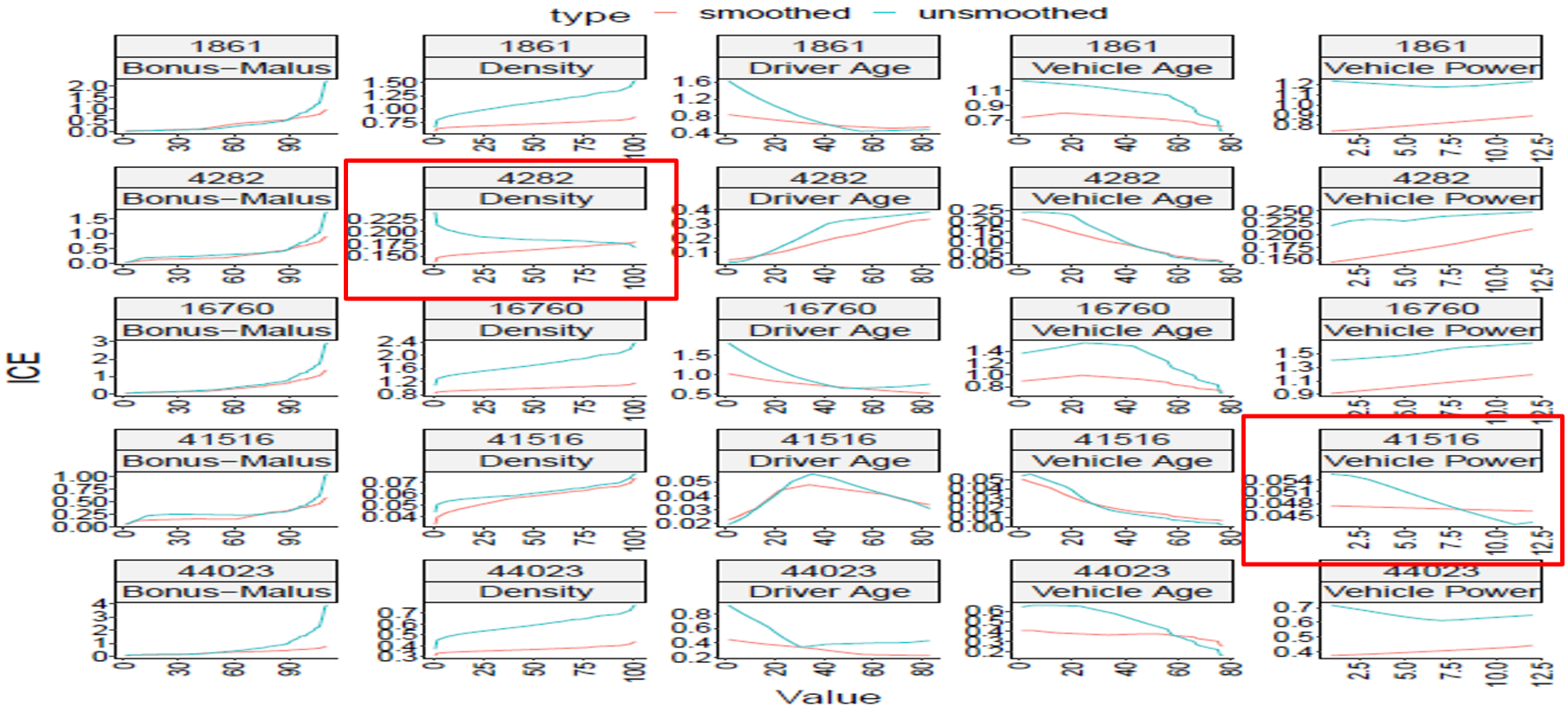


Local ICEnet – did it work?

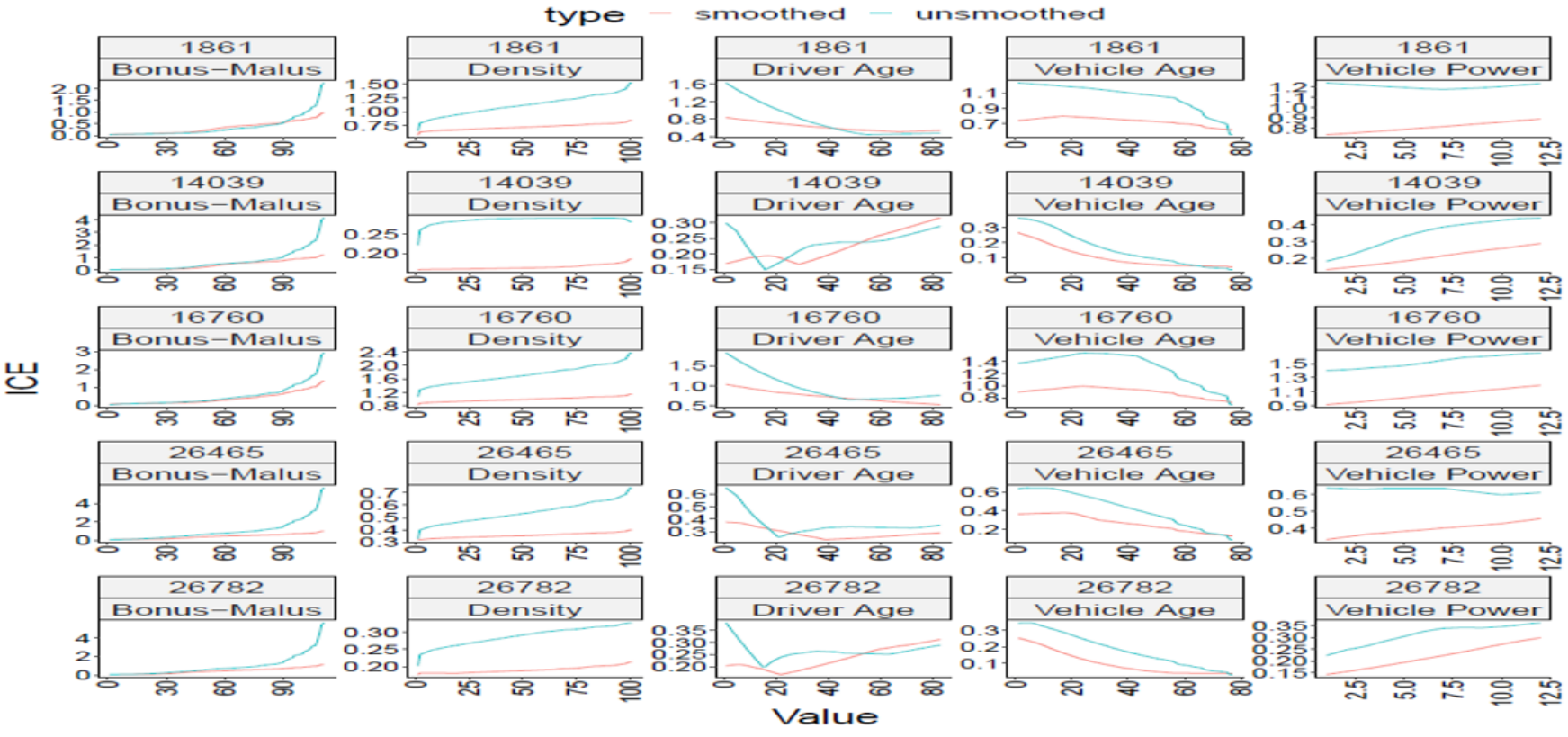
- Moderately successful at enforcing required constraints
- Most of the variables show improved monotonicity and smoothness scores...
- ...except for bonus-malus level and density covariates.
- Could vary the size of the window as a hyper-parameter



Local ICEnet – monotonicity results



Local ICEnet – smoothness results



Closing thoughts

- Presented a novel approach - the ICEnet - to constrain neural network predictions to be monotonic and smooth
- Connected to ICE model interpretability technique
- Global ICEnet can be approximated using less computationally intensive version = Local ICEnet
- Fitting these models to real-world open-source dataset has shown:
  - monotonicity constraints enforced when fitting ICEnet can improve out-of-sample performance of actuarial models
  - models with a combination of smoothing and monotonic constraints allows models to produce predicted frequencies of claim that accord with intuition and are commercially reasonable
- Could also impose constraints on the global behaviour of networks instead i.e. on PDPs
- Other formulations of smoothing constraints could be imposed e.g. absolute differences could be used

Discussion points

- Do we need smoothness constraints for non-life pricing?
- Balance between predictive accuracy (jaggedness) and credibility
- Is a fusion constraint a better approach?
- Can we verify a model is "working" through output constraints such as ICEnet?
- What other constraints need to be introduced for actuarial modelling?
- See paper for full references

References I

Arora, S., Cohen, N., Hu, W., & Luo, Y. (2019). Implicit regularization in deep matrix factorization. In *Neurips*. Retrieved from https://arxiv.org/abs/1905.13655

aw Jastrzębski, S., et al. (2018). *The width of minima reached by stochastic gradient descent is influenced by learning rate to batch size ratio*. Retrieved from https://www.research.ed.ac.uk/files/75846467/width_of_minima_reached_by_stochastic_gradient_descent_is_influenced_by_learning_rate_to_batch_size_ratio.pdf (preprint)

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer. (See §3.2, pp. 147–150, eq. (3.41))

Gunasekar, S., Woodworth, B. E., Bhojanapalli, S., Neyshabur, B., & Srebro, N. (2017). Implicit regularization in matrix factorization. In *Neurips*. Retrieved from https://papers.nips.cc/paper/7195-implicit-regularization-in-matrix-factorization

References II

Hastie, T., Tibshirani, R., & Wainwright, M. (2015). *Statistical learning with sparsity: The lasso and generalizations*. Chapman and Hall/CRC.

Hochreiter, S., & Schmidhuber, J. (1997). Flat minima. *Neural Computation*, *9*(1), 1–42. Retrieved from https://direct.mit.edu/neco/article/9/1/1/6027/Flat-Minima

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In *International conference on learning representations (iclr)*. Retrieved from https://arxiv.org/abs/1609.04836

Li, H., Xu, Z., Taylor, G., Studer, C., & Goldstein, T. (2018). Visualizing the loss landscape of neural nets. In *Neurips*. Retrieved from https://papers.neurips.cc/paper/7875-visualizing-the-loss-landscape-of-neural-nets.pdf

References III

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=Bkg6RiCqY7

Mandt, S., Hoffman, M. D., & Blei, D. M. (2017). Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research*, *18*(134), 1–35. Retrieved from https://www.jmlr.org/papers/volume18/17-214/17-214.pdf

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F. A., ... Courville, A. (2019). On the spectral bias of neural networks. In *Proceedings of the 36th international conference on machine learning (icml)*. Retrieved from https://proceedings.mlr.press/v97/rahaman19a/rahaman19a.pdf

Richman, R., & Wüthrich, M. V. (2024). Smoothness and monotonicity constraints for neural networks using icenet. *Annals of Actuarial Science*, *18*(3), 712–739. doi: 10.1017/S174849952400006X

References IV

Smith, S. L., Elsen, E., & De, S. (2020). On the generalization benefit of noise in stochastic gradient descent. In *Proceedings of the 37th international conference on machine learning (icml)*. Retrieved from https://proceedings.mlr.press/v119/smith20a/smith20a.pdf

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, *58*(1), 267–288.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., & Knight, K. (2005). Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B*, *67*(1), 91–108.

Tikhonov, A. N., & Arsenin, V. Y. (1977). *Solutions of ill-posed problems*. Winston.

References V

Valle-Pérez, G., Camargo, C. Q., & Louis, A. A. (2019). Deep learning generalizes because the parameter-function map is biased toward simple functions. In *International conference on learning representations (iclr).* Retrieved from https://cclab.science/papers/ICLR_2019.pdf

Wager, S., Wang, S., & Liang, P. (2013). Dropout training as adaptive regularization. In *Advances in neural information processing systems.*

Yuan, M., & Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B*, *68*(1), 49–67.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B*, *67*(2), 301–320.