AI for Statistical Analysis

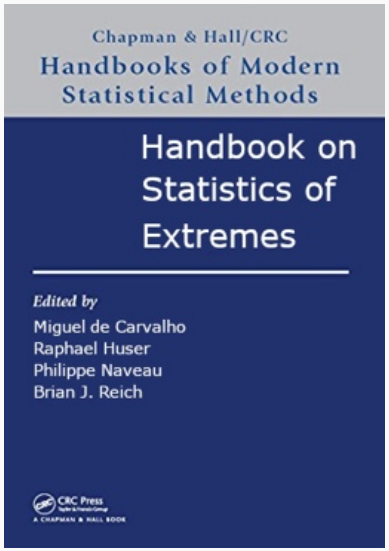
Reetam Majumder, University of Arkansas
ft. Neil Kpamegan, University of Maryland Baltimore County

UMBC | Oct 31, 2025

Outline

- Part I: Basics of deep learning. Background, foundations, rules of thumb.
- Part II: Using deep learning for statistical analysis. Linear/non-linear regression, quantile regression, density estimation, parameter estimation and simulation-based inference.
- Part III: Further topics. Assessing performance, ongoing research, and resources.

Outline



- Much of the content in parts I and II is based on an AI and extremes short course that Jordan Richards (Edinburgh), Likun Zhang (Mizzou) and I ran during the Contemporary Advances in Statistics of Extremes workshop held at Mizzou in June 2025.
- Deep learning has seen a lot of adoption in the extremes community (keep an eye out for the upcoming handbook!)
- Code and slides for this course at `https://github.com/reetamm/AI4stats`.

Learning goals

Topics I will cover in code and in slides:

- Why and how to do regression using deep learning
- Density estimation
- Parameter estimation and simulation-based inference

Topics I will cover only in slides:

- Uncertainty quantification, explainability, interpretability
- Some theoretical work
- State of the science

Topics I will not cover:

- Advanced frameworks like transformers, U-nets, PINNs
- Reinforcement learning, causal inference, generative AI

Disclaimers

1. This isn't a comprehensive tutorial by any means. The field is advancing really fast.
   This is just a starter pack for statistical estimation/inference.

2. I've not offered this before, so I don't know how long this will take. We will play it by ear.
   I have office hours tomorrow at MP401 from 12-3 PM.

Preliminaries: Deep learning packages

- The two most popular deep learning packages are `tensorflow` (Abadi et al., 2015) and `torch` (Paszke et al., 2017)
- `tensorflow` was written in Python, while `torch` was written in `C++` with the python version known as `pytorch`
- For the most part, their development happens on `Python`, and hundreds (if not more?) packages use them as their basis
- `tensorflow` has a higher level API called `keras`, which simplifies a lot of the technical aspects.
- `torch` has something similar, called `luz`.
- Both `tensorflow/keras` and `torch/luz` are available on `R`, but their implementations couldn't be more different.
- `tf` on `R` needs `Python`, while `torch` doesn't.

keras vs. torch on R

	`tf/keras`	`torch`
Ease of installation	Complicated*	Straightforward
Features	Several Python packages available through `reticulate`	Limited to packages developed on R
Accessibility	keras makes things easy	Slowly getting better
GPU support	Yes	Yes
Overhead	High on personal systems, but easy to run on Google Colab	Harder to run on Colab, but easy on personal devices

- `tf/keras` on `R` is just a wrapper of the `Python` library made possible by `reticulate`
- Since Google Colab already has Python, it bypasses much of the difficulty and allows the use of `keras3` with relative ease.
- *`keras3` is much easier to install on local machines
- Google Colab does not support the older `keras` package any more.

Part I: Basics of deep learning

Regression: General Setting

Given:

- Response variable Y (often in $\mathbb{R}$, but could also be discrete classes);
- Covariates $X \in \mathbb{R}^q$.

We are interested in learning some **estimand** that describes the conditional distribution of $Y \mid \mathbf{X} = \mathbf{x}$.

Typical estimands of interest are:

- Class probabilities: $\Pr\{Y = k|\mathbf{X} = \mathbf{x}\}$ for $k \in \mathbb{N}$;
- Expectation: $\mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$;
- Quantiles: $Q_x(\tau) = \inf\{y : F_{Y|\mathbf{X}}(y|x) \geq \tau\}$ for $\tau \in (0, 1)$;
- "Parameters" of $F_{Y|\mathbf{X}}$. Can include parameters in the standard finite-dimensional sense, but may also be interested in semi-/non-parametric model for $F_{Y|\mathbf{X}}$.

Regression: How do we estimate these parameters?

Set up an estimable model $\mathbf{m} : \mathbb{R}^q \mapsto \boldsymbol{\Theta}$:

- Let $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ contain your estimands of interest.
- Setup a function $\mathbf{m}$ that maps your covariates $\mathbf{x}$ to $\boldsymbol{\theta}$, i.e., $\boldsymbol{\theta}(\mathbf{x}) := \mathbf{m}(\mathbf{x})$.
- Get yourself some data $\{(y_i, \mathbf{x}_i)\}_{i=1}^n$.

Estimate $\mathbf{m}$ via minimisation of some loss or cost function:

- Class probabilities: binary cross-entropy;
- Expectation: MSE $n^{-1} \sum_{i=1}^n (y_i - \theta(\mathbf{x}_i))^2$
- Quantiles: Pinball loss $n^{-1} \sum_{i=1}^n (y_i - \theta(\mathbf{x}_i))(\tau - \mathbb{1}\{y_i < \theta(\mathbf{x}_i)\})$;
- $F_{Y|\mathbf{x}}$. Associated negative log-likelihood.

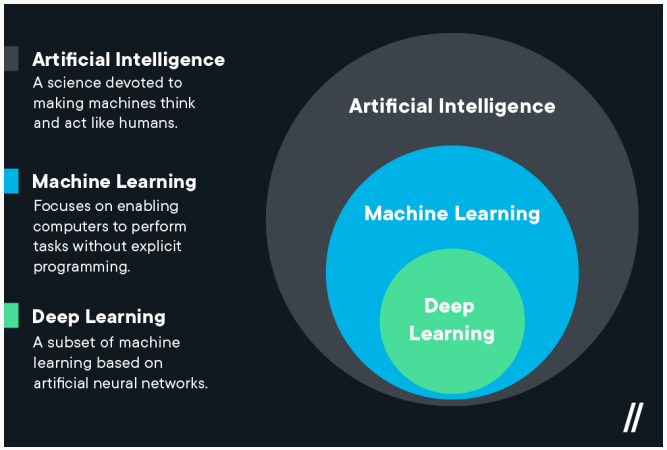
For example, an estimate for the conditional expectation function, say $\widehat{\theta(\mathbf{x})}$, is the minimiser

$$\widehat{\theta(\mathbf{x})} \in \arg\min_{\mathbf{m}\in\mathcal{M}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{m}(\mathbf{x}_i))^2,$$

where $\mathcal{M}$ is some appropriate space of estimable functions.

But what should $\mathcal{M}$ look like?

Machine learning



- The **fundamental** difference between statistics and AI/ML/DL is the complexity of $\mathbf{m}(\cdot)$.
- Traditional statistics want $\mathbf{m}(\cdot)$ simple, e.g., linear, parametric, splines, so that estimates are interpretable.
- ML/DL wants flexibility in $\mathbf{m}(\cdot)$.
- Other key ideas - optimisation, regularisation, testing - tend to be the same.

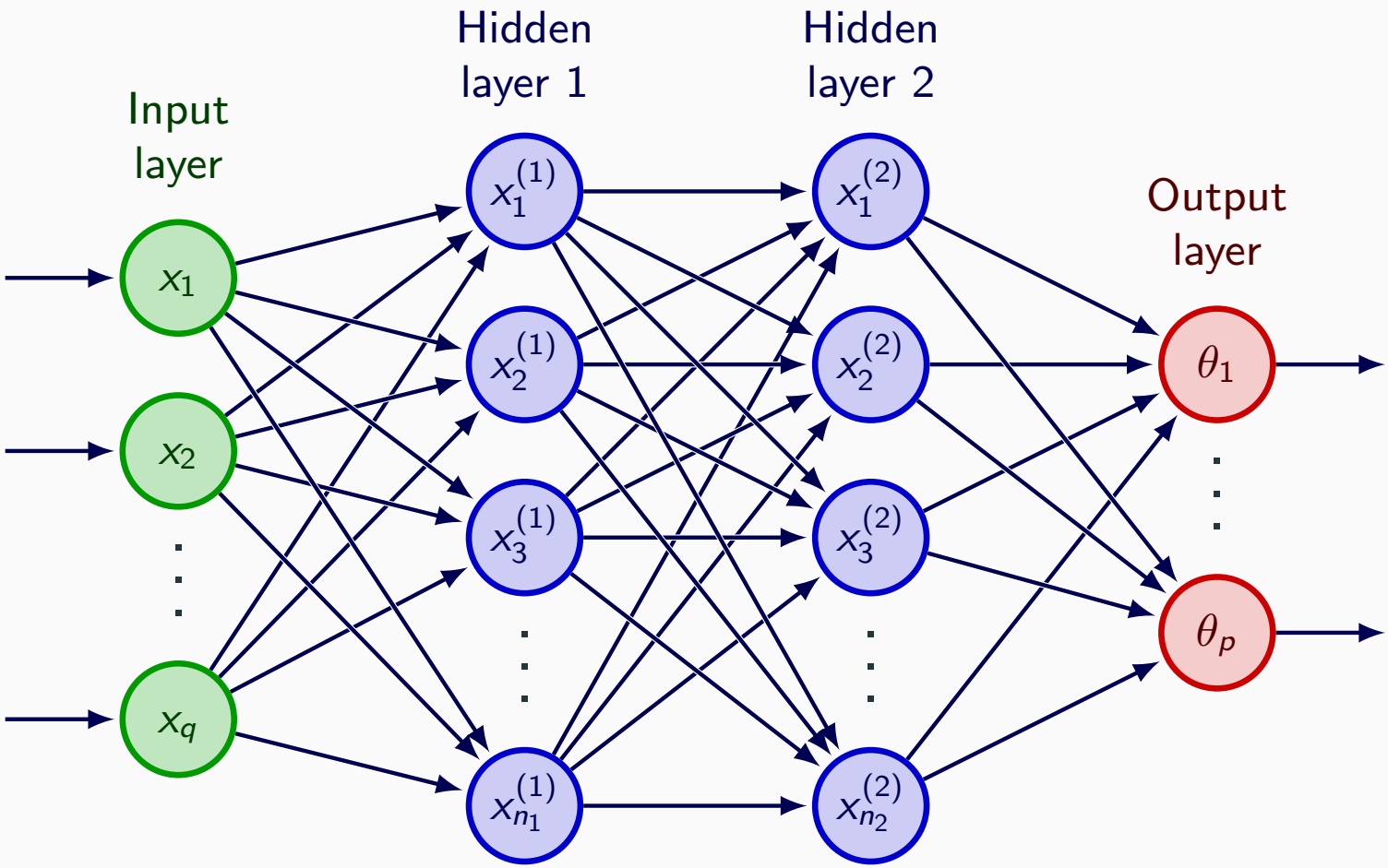
Copyright belongs to `https://flatironschool.com/blog/deep-learning-vs-machine-learning/`

Deep learning

The **deep** in Deep learning comes from the depth of **m**:

- We represent **m** as a neural network. This is just a composition of **layers** of simple, differentiable operations.
- There are lots of different choices of operations, which give rise to different types of neural network. Today, we focus on the standard feed-forward multi-layered perceptron

Deep learning



Multi-layer Perceptron (MLP)

The neural network $\mathbf{m}_\psi$ is constructed as a composition of $L \in \mathbb{N}$ *hidden layers*, $\mathbf{m}^{(l)}$ for $l = 1, \ldots, L$, and an *output/final* layer, $\mathbf{m}^{(L+1)}$, such that $\mathbf{m}_\psi(\cdot) := \mathbf{m}^{(L+1)} \circ \cdots \circ \mathbf{m}^{(1)}(\cdot)$.

- ψ contains all the estimable parameters (weights and biases);
- the neural network has the hierarchical structure

$$
\begin{aligned}
\mathbf{x}^{(1)} &:= \mathbf{m}^{(1)}(\mathbf{x}) = \mathbf{a}^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right), \\
\mathbf{x}^{(2)} &:= \mathbf{m}^{(2)}(\mathbf{x}^{(1)}) = \mathbf{a}^{(2)}\left(\mathbf{W}^{(2)}\mathbf{x}^{(1)} + \mathbf{b}^{(2)}\right), \\
&\vdots \\
\mathbf{x}^{(L)} &:= \mathbf{m}^{(L)}(\mathbf{x}^{(L-1)}) = \mathbf{a}^{(L)}\left(\mathbf{W}^{(L)}\mathbf{x}^{(L-1)} + \mathbf{b}^{(L)}\right), \\
\theta(\mathbf{x}) &:= \mathbf{m}^{(L+1)}(\mathbf{x}^{(L)}) = \mathbf{a}^{(L+1)}\left(\mathbf{W}^{(L+1)}\mathbf{x}^{(L)} + \mathbf{b}^{(L+1)}\right).
\end{aligned}
$$

Activation functions

Activation functions provide flexibility by making the layers non-linear.

Popular choices includes:

- ReLU: $\mathbf{a}(\mathbf{x})_j = \max(x_j, 0)$
- sigmoid: $\mathbf{a}(\mathbf{x})_j = \exp(x_j)/(1 + \exp(x_j))$
- tanh
- eLU and leaky ReLU
- softmax: $\mathbf{a}(\mathbf{x}) = (\frac{\exp(x_1)}{\sum_i \exp(x_i)}, \frac{\exp(x_2)}{\sum_i \exp(x_i)}, \dots)$

Some common layer types

A list of the different layer types can be found at `https://keras3.posit.co/reference/index.html`. Some interesting ones are:

- Reshaping: flatten, reshape, upsampling, zero-padding
- Pooling: average pooling, max pooling
- Convolution: These are used to process image data
- Temporal: recurrent, gated recurrent unit (GRU), long short-term memory (LSTM)
- Attention: `https://research.google/pubs/attention-is-all-you-need/`
- Misc.: batchnorm, dropout, add, average, concatenate....

Training models

Models are trained by solving the empirical risk minimisation problem

$$\widehat{\boldsymbol{\psi}} \in \arg\min_{\boldsymbol{\psi}} \frac{1}{n} \sum_{i=1}^{n} \ell\{y_i; \mathbf{m}_{\boldsymbol{\psi}}, (\mathbf{x}_i)\},$$

where ℓ is our loss function.

In **statistics**, this would be achieved using gradient descent, via the update

$$\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \frac{\lambda}{n} \sum_{i=1}^{n} \boldsymbol{\nabla}_{\boldsymbol{\psi}} \ell\{y_i, \mathbf{m}_{\boldsymbol{\psi}}(\mathbf{x}_i)\}, \tag{1}$$

where $\lambda > 0$ is a tunable learning rate and $\boldsymbol{\nabla}_{\boldsymbol{\psi}}$ denotes the differential operator with respect to entries in $\boldsymbol{\psi}$.

Training deep models

In **deep learning**, where n is typically large, a more economical approach is to update over individual observations (SGD), using the rule

$$\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \lambda \boldsymbol{\nabla}_{\boldsymbol{\psi}} \ell\{y_i, \mathbf{g}_{\boldsymbol{\psi}}(\mathbf{x}_i)\}, \qquad i = 1, \ldots, n. \tag{2}$$

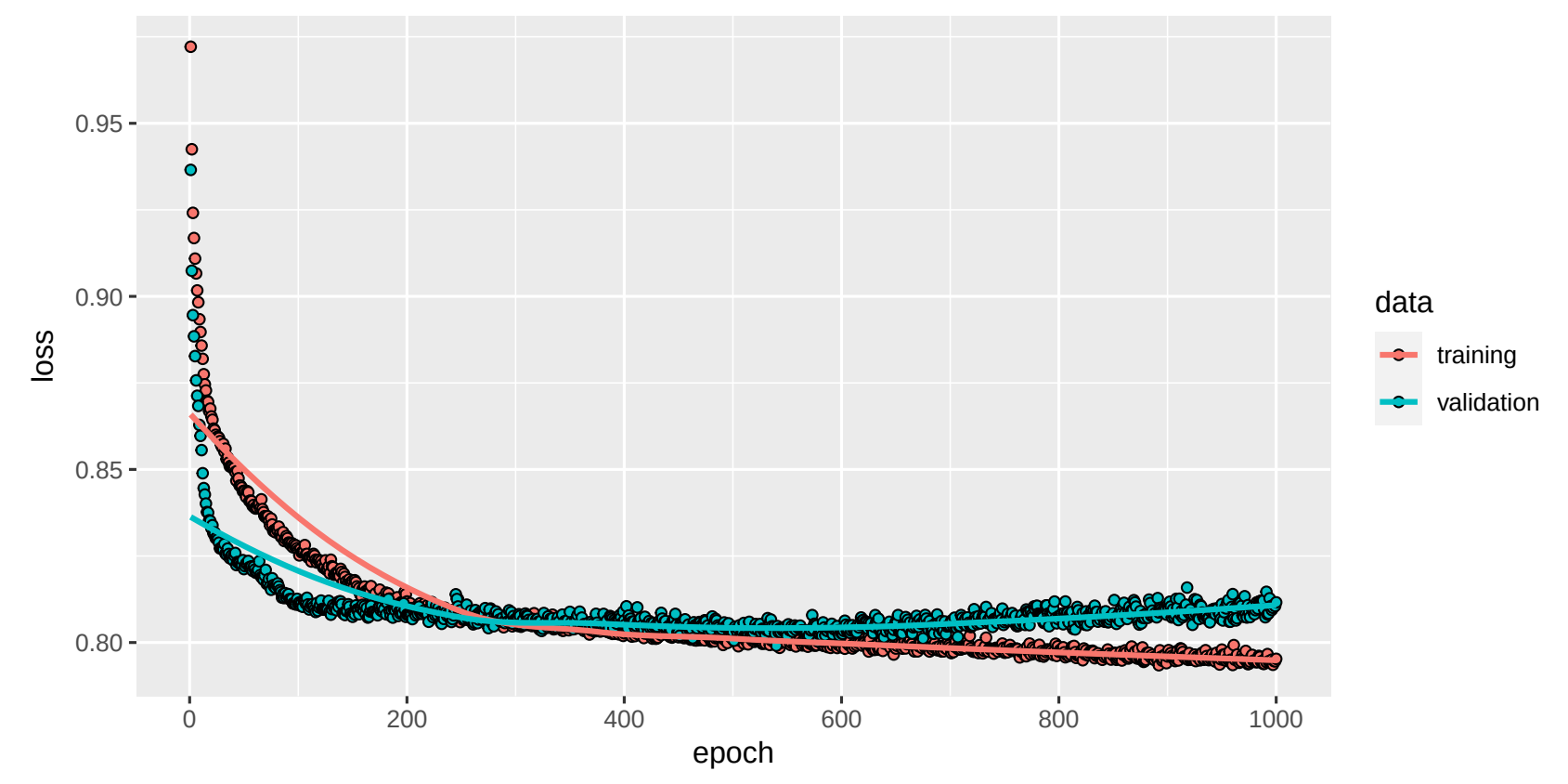
Neural networks are usually trained using an algorithm that falls in-between gradient descent and true SGD, which is called *mini-batch SGD*. In this case, all samples are split into n_b mini-batches of equal size, and n_b updates are performed for every **epoch** of the training scheme.

Usually, $\boldsymbol{\nabla}_{\boldsymbol{\psi}}$ is tractable and fast-to-compute via **backpropagation**. This is because the neural network model is a composition of simple differentiable operations.

Regularisation: Training/validation/testing

Due to their large number of parameters, neural networks are prone to overfitting. It's necessary to use **validation/testing** to assess overfitting.

- Subset the data into training, validation, and testing.
- Each set has a different purpose:
  - The model is **trained** on the training data. This is used to update the weights and biases.
  - We **validate** the model on the validation data. Typically used for model selection.
  - To get a truly unbiased evaluation of the model fit and to compare amongst different models, we **test** the model on test data.


0.95
0.90
0.85
0.80
loss
0
200
400
600
800
1000
epoch
data
training
validation

Regularisation/numerics

Other forms of regularisation to consider:

- Weight penalties (think LASSO or ridge)
- Random weight initialization
- Dropout
- Sharing weights
- Early-stopping and checkpoints

Some numerical aspects:

- Pre-training
- Standardising input data
- Parameter dependent support
- Optimizing architecture/training scheme (please don't ask about this)

Part II: Deep Learning for Statistical Analysis

Regression

Regression problems

The most obvious use of these models are for supervised learning for a continuous response variable, viz, regression.

We will look at three regression examples:

- Deep regression - A neural network that predicts a continuous response while minimizing mean squared error loss, effectively targeting the expected value.
- Linear regression - A shallow neural network with a single hidden unit and a linear activation function and mean squared error loss, equivalent to a linear model.
- Quantile regression - A neural network that targets the $\tau = 0.85$ quantile and uses the pinball loss.

Density estimation

Estimating entire densities

- In statistics, we usually want to make probability statements based on a model fitted to the data
- For example, we might assume that:

$$Y_i|X_i \sim Normal(\mu(X_i), \sigma^2(X_i))$$

- We could then train a neural network to output $(\mu(X_i), \sigma^2(X_i))$ pairs while maximizing the log-likelihood of the Normal distribution.
- However, sometimes we *don't* want to assume a parametric form of the density
- In that case, we need:
  - A non/semi-parametric way to represent the density (conditional or marginal)
  - A loss function which evaluates the (usual negative log-likelihood) loss between the response y_i and the estimates $\pi(x_i)$

Semi-parametric quantile regression

SPQR (Xu and Reich, 2021) represents a continuous distribution $Y|X$ as:

$$f(y|x) = \sum_{k=1}^{K} \pi_k(x) M_k(y)$$

- $\{\pi_k\}_{k=1}^{K}$ is a vector of probabilities output from a NN
- The loss function is $-\log f(y|x)$
- $M-$splines are rescaled $B-$splines
- $I-$splines are just integrated $M-$splines such that:

$$F(y|x) = \sum_{k=1}^{K} \pi_k(x) I_k(y)$$

- Representation is scale-invariant

Simulation-based inference

Intractable models and/or massive datasets

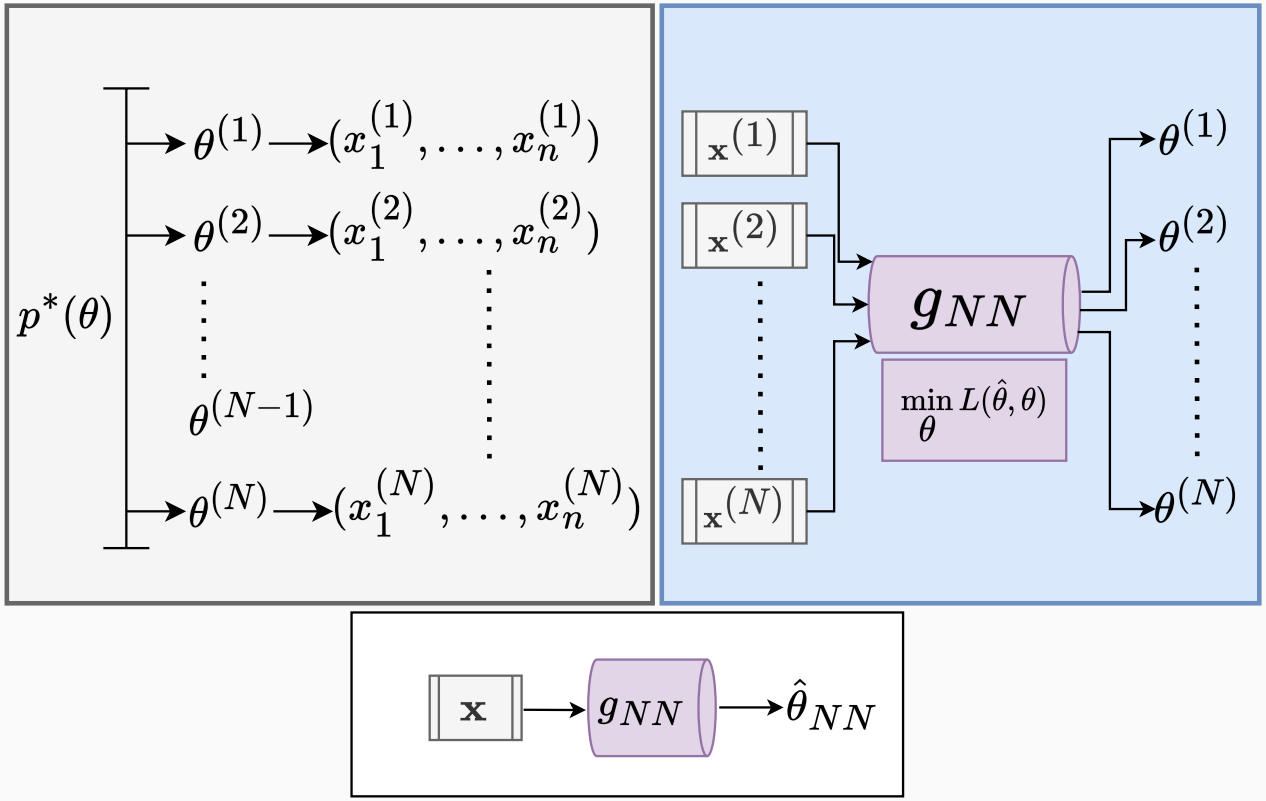
- A lot of modern statistics deals with massive data
- Doing likelihood-based inference for these models can be challenging because often they are computationally expensive to evaluate
- Example 1: Gaussian processes have a computational complexity of $\mathcal{O}(n^3)$, where n is the number of observations (or spatial locations) and memory requirements of $\mathcal{O}(n^2)$
- This was a bottleneck a decade back before scalable methods came along
- Still challenging to do exact inference on more than a few thousand locations on your laptop
- Example 2: The max stable process is a spatial extremes model that was (and remains) quite popular
- The number of terms in the likelihood is a Bell number
- E.g., when $n = 10$ one would need to sum up around 116000 terms to compute the contribution of a single observation to the likelihood

Simulating from complex likelihoods

- However, for many of these models, sampling from the model is often inexpensive and fast
- E.g., sampling from a GP just needs matrix multiplication, whereas evaluating the likelihood requires matrix inversions
- Enter simulation-based inference

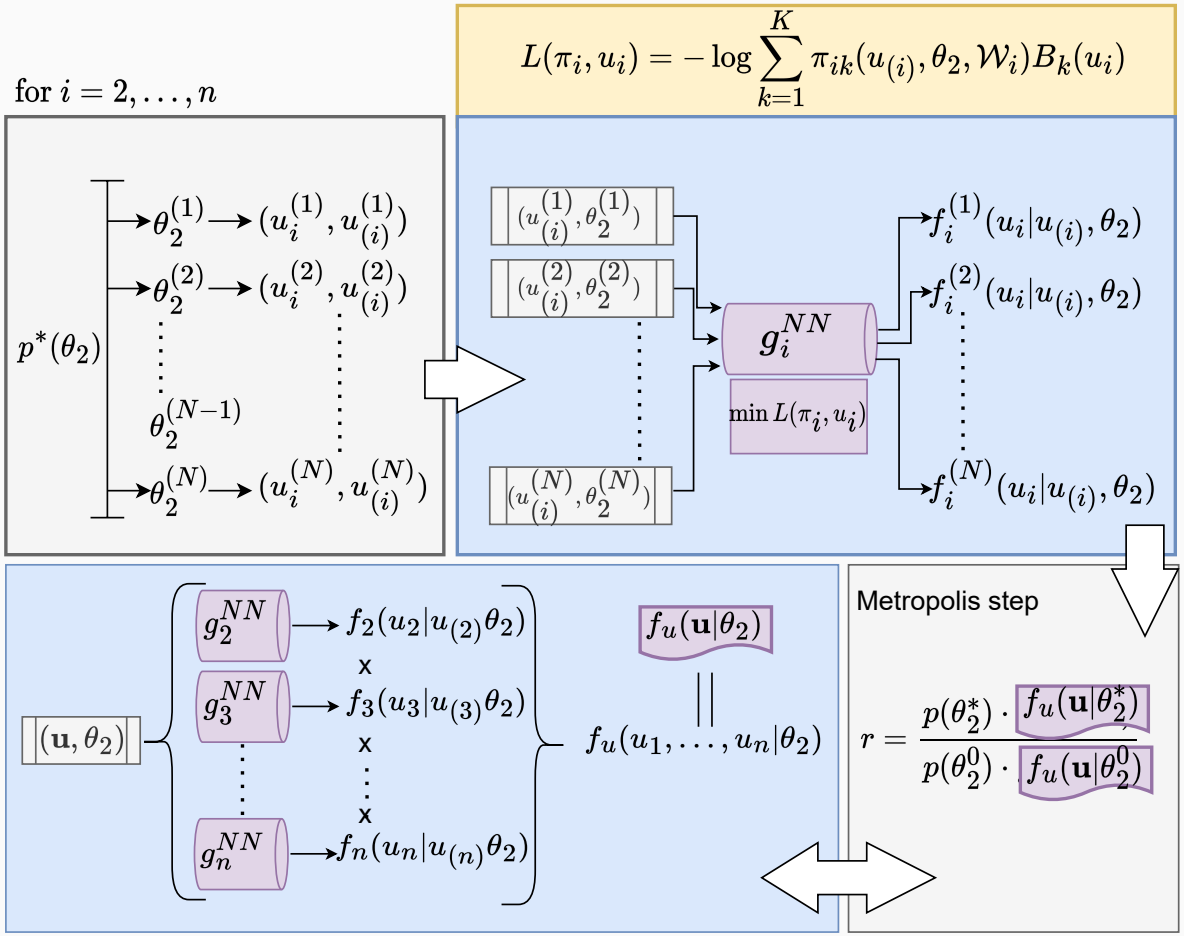
Goal: Learn a mapping from data to a functional of the parameters (usually a frequentist or pesudo-Bayesian estimate of the parameter, or sometimes a surrogate likelihood) using a neural network, trained on a bunch of simulated data that are cheap to generate.

Neural estimation for parameters



This is by far the most common framework for simulation-based inference (e.g., Banesh et al., 2022; Gerber and Nychka, 2021), and is the version we will try out in code.

Neural estimation of likelihood surfaces



This is harder to do, but more statistical machinery is available by the end of it. This particular pipeline was used in Majumder et al. (2024).

Part III: Further topics

Uncertainty, interpretability, and xAI

Uncertainty quantification (UQ)

- Any modeling exercise contends with several sources of uncertainty
- Let me start with the uncertainty associated with the NN architecture
- In all the cases we have talked about, given the data and architecture, the uncertainty arises from:
  - The initialization of the weights
  - The stochastic gradient descent algorithm
  - The internal train-validation split
- Setting a seed will fix these
- However, a more robust solution might be to train an ensemble of models
  - You could initialize the training weights yourself (differently in each model)
  - You can manually supply the train and validation sets instead of letting the software do it
  - You could train different models with the same seed either with subsamples (cross-validation) or with bootstrapped data
- This is not exhaustive! Deep learning experts can give you more ideas on how to do useful UQ

Explainable AI (xAI)

- The main goal of xAI (at least in this statistical context) is to explain how the inputs (covariates) affect the outputs - a.k.a. feature importance
- There are local metrics like LIME, Shapley values, and SHAP, which quantify how covariates (features) affect individual predictions
- There are also global metrics like ALE (Apley and Zhu, 2020) and GOALS (Winn-Nuñez et al., 2024) which quantify how features affect the entire distribution of predictions
- The aforementioned methods are largely model agnostic and can be used for any black-box model. Wikle et al. (2023) is a great recent resource discussing these in a statistical context
- A lot of the machine learning literature group these under interpretability, but as a statistician, there is a distinction to be made. Molnar (2025) is an accessible book which covers several of these methods

Bayesian neural networks

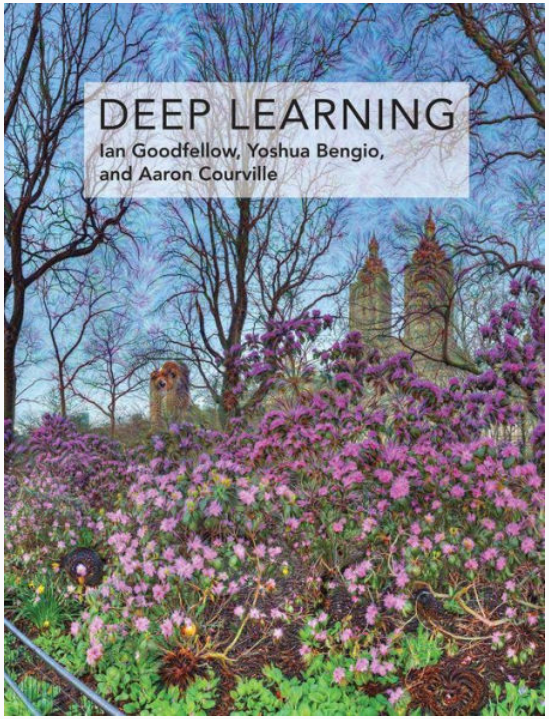
- The classical approach here is to put priors on the weights and biases
- Common priors include GP, automatic relevance determination (ARD), and Gaussian scale mixtures (GSM)
- Instead of optimization, the weights are learnt via sampling; Hamiltonian Monte-Carlo (HMC) and no U-turns sampler (NUTS) is common
- R packages I am aware of - bnns, BayesFluxR, SPQR

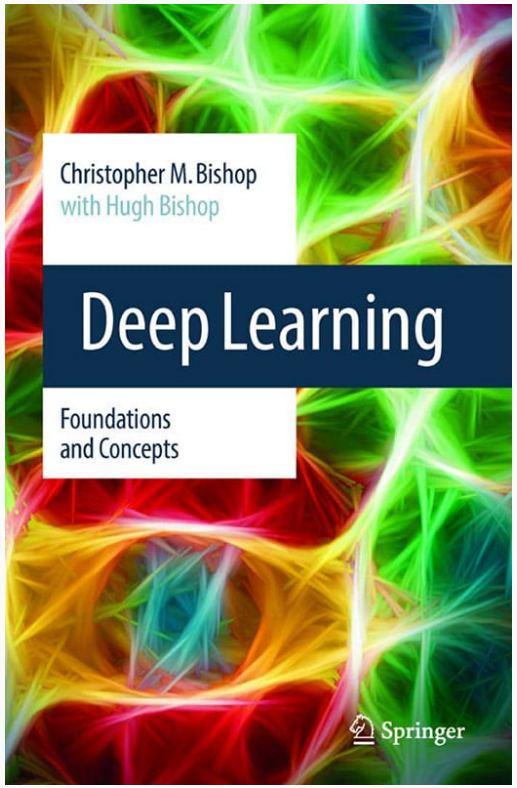
There is one alternative methodology I am aware of:

- Gal and Ghahramani (2016) suggested using dropout layers as a form of quantifying Bayesian uncertainty
- They argue that a dropout layer applied before every weight layer makes an arbitrary neural network mathematically equivalent to an approximation to the probabilistic deep Gaussian process

Further reading

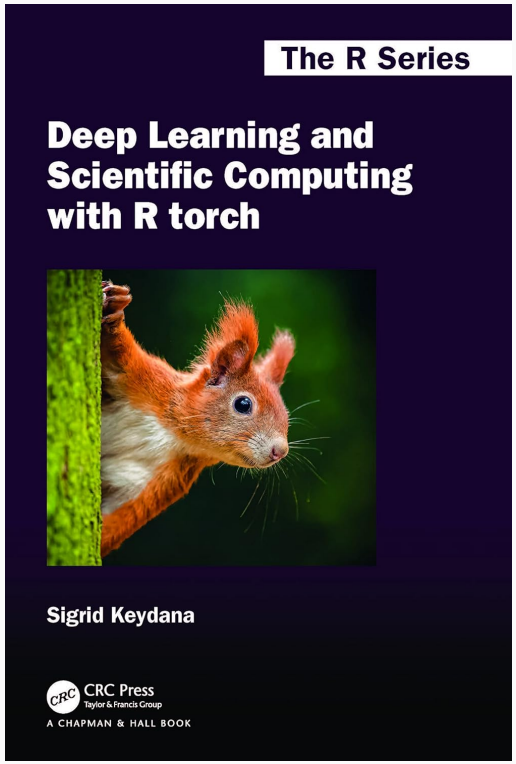
Textbooks on deep learning (methods)

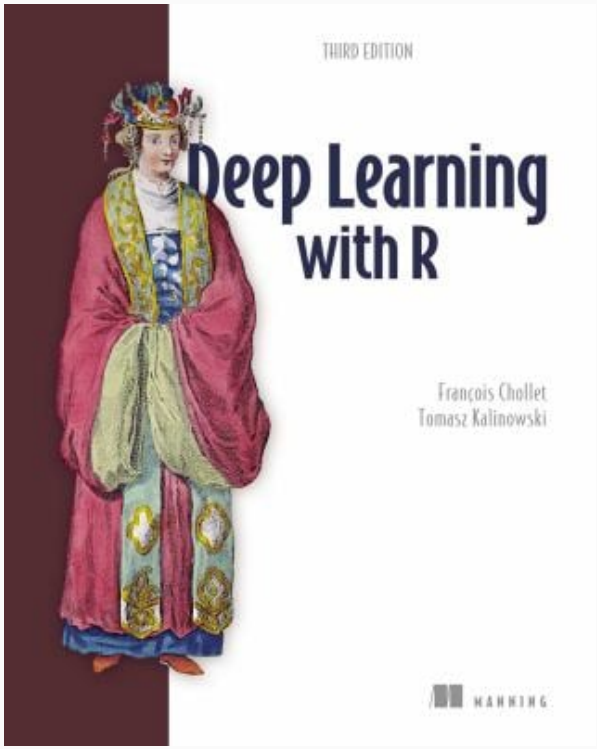




- `https://www.bishopbook.com/` - Bishop and Bishop (2023)
- `https://www.deeplearningbook.org/` - Goodfellow et al. (2016)
- Both have free online versions on their websites

Textbooks on deep learning (computation)





- Keydana (2023) for `torch`
- Chollet and Allaire (2025) for `keras`

Some relevant R packages

Modeling:

- `deeptrafo` (Kook et al., 2024) - deep density regression with multimodal input support. CRAN URL.
- `SPQR` (Xu et al., 2022) - torch package for the methodology discussed today. GitHub URL.
- `NeuralEstimators` (Sainsbury-Dale et al., 2024a) - Parameter estimation via simulation-based inference. CRAN URL.
- `tabnet` - attentive interpretable tabular learning (Arik and Pfister, 2020). CRAN URL.

Feature importance:

- Several packages for Shapley values, SHAP etc.
- `ale` for accumulated local effects. CRAN URL
- GOALS has a GitHub repository with code (link). I've not tested it though.

Special topics

Fast inference:

- Neural estimation:
  - Spatio-temporal: (Lenzi et al., 2023; Sainsbury-Dale et al., 2024a,b; Richards et al., 2024; Sainsbury-Dale et al., 2025; Dell'Oro and Gaetan, 2024; Rai et al., 2025; Hector and Lenzi, 2024)
  - Multivariate: (André et al., 2025; Hua, 2025)
  - Univariate: (Rai et al., 2024; Richards et al., 2025)
- Intractable likelihood approximation: (Majumder and Reich, 2023; Majumder et al., 2024; Walchessen et al., 2024)
- Variational inference: (Maceda et al., 2024)
- Neural networks for geospatial data (Zhan and Datta, 2025)

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

André, L. M., Wadsworth, J. L., and Huser, R. (2025). Neural bayes inference for complex bivariate extremal dependence models. *arXiv preprint arXiv:2503.23156*.

Apley, D. W. and Zhu, J. (2020). Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82:1059–1086.

Arik, S. O. and Pfister, T. (2020). Tabnet: Attentive interpretable tabular learning.

References ii

Banesh, D., Panda, N., Biswas, A., Roekel, L. V., Oyen, D. A., Urban, N. M., Grosskopf, M., Wolfe, J., and Lawrence, E. C. (2022). Fast gaussian process estimation for large-scale in situ inference using convolutional neural networks. Los Alamos National Laboratory (LANL), Los Alamos, NM (United States).

Bishop, C. M. and Bishop, H. (2023). *Deep Learning - Foundations and Concepts*. 1st edition.

Chollet, F. and Allaire, J. J. (2025). *Deep Learning with R*. Manning Publications Co., USA, 3rd edition.

Dell'Oro, L. and Gaetan, C. (2024). Flexible space-time models for extreme data. *arXiv preprint arXiv:2411.19184*.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1050–1059. JMLR.org.

Gerber, F. and Nychka, D. (2021). Fast covariance parameter estimation of spatial gaussian process models using neural networks. *Stat*, 10(1):e382.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

References iii

Hector, E. C. and Lenzi, A. (2024). When the whole is greater than the sum of its parts: Scaling black-box inference to large data settings through divide-and-conquer. *arXiv preprint arXiv:2412.20323*.

Hua, L. (2025). Amortized neural inference on bivariate tail dependence and tail asymmetry. *Available at SSRN 5287687*.

Keydana, S. (2023). *Deep Learning and Scientific Computing with R torch*. CRC Press.

Kook, L., Baumann, P. F. M., Dürr, O., Sick, B., and Rügamer, D. (2024). Estimating conditional distributions with neural networks using R package deeptrafo. *Journal of Statistical Software*, 111(10):1–36.

Lenzi, A., Bessac, J., Rudi, J., and Stein, M. L. (2023). Neural networks for parameter estimation in intractable models. *Computational Statistics & Data Analysis*, 185:107762.

Maceda, E., Hector, E. C., Lenzi, A., and Reich, B. J. (2024). A variational neural bayes framework for inference on intractable posterior distributions. *arXiv preprint arXiv:2404.10899*.

Majumder, R. and Reich, B. J. (2023). A deep learning synthetic likelihood approximation of a non-stationary spatial model for extreme streamflow forecasting. *Spatial Statistics*, 55:100755.

References iv

Majumder, R., Reich, B. J., and Shaby, B. A. (2024). Modeling extremal streamflow using deep learning approximations and a flexible spatial process. *The Annals of Applied Statistics*, 18(2):1519–1542.

Molnar, C. (2025). *Interpretable Machine Learning*. 3 edition.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Rai, S., Hoffman, A., Lahiri, S., Nychka, D. W., Sain, S. R., and Bandyopadhyay, S. (2024). Fast parameter estimation of generalized extreme value distribution using neural networks. *Environmetrics*, 35(3):e2845.

Rai, S., Nychka, D. W., and Bandyopadhyay, S. (2025). Modeling spatial extremes using non-gaussian spatial autoregressive models via convolutional neural networks. *arXiv preprint arXiv:2505.03034*.

Richards, J., Alotaibi, N., Cisneros, D., Gong, Y., Guerrero, M. B., Redondo, P. V., and Shao, X. (2025). Modern extreme value statistics for utopian extremes. eva (2023) conference data challenge: Team yalla. *Extremes*, 28(1):149–171.

Richards, J., Sainsbury-Dale, M., Zammit-Mangion, A., and Huser, R. (2024). Likelihood-free neural Bayes estimators for censored inference with peaks-over-threshold models. *Journal of Machine Learning Research*, 25(390):1–49.

References

Sainsbury-Dale, M., Zammit-Mangion, A., Cressie, N., and Huser, R. (2025). Neural parameter estimation with incomplete data. *arXiv preprint arXiv:2501.04330*.

Sainsbury-Dale, M., Zammit-Mangion, A., and Huser, R. (2024a). Likelihood-free parameter estimation with neural Bayes estimators. *The American Statistician*, 78(1):1–14.

Sainsbury-Dale, M., Zammit-Mangion, A., Richards, J., and Huser, R. (2024b). Neural bayes estimators for irregular spatial data using graph neural networks. *Journal of Computational and Graphical Statistics*, (just-accepted):1–28.

Walchessen, J., Lenzi, A., and Kuusela, M. (2024). Neural likelihood surfaces for spatial processes with computationally intensive or intractable likelihoods. *Spatial Statistics*, 62:100848.

Wikle, C. K., Datta, A., Hari, B. V., Boone, E. L., Sahoo, I., Kavila, I., Castruccio, S., Simmons, S. J., Burr, W. S., and Chang, W. (2023). An illustration of model agnostic explainability methods applied to environmental data. *Environmetrics*, 34(1):e2772.

Winn-Nuñez, E. T., Griffin, M., and Crawford, L. (2024). A simple approach for local and global variable importance in nonlinear regression models. *Computational Statistics & Data Analysis*, 194:107914.

References vi

Xu, S. G., Majumder, R., and Reich, B. J. (2022). SPQR: An R package for semi-parametric density and quantile regression. https://arxiv.org/abs/2210.14482.

Xu, S. G. and Reich, B. J. (2021). Bayesian nonparametric quantile process regression and estimation of marginal quantile effects. *Biometrics*, 79(1):151–164.

Zhan, W. and Datta, A. (2025). Neural networks for geospatial data. *Journal of the American Statistical Association*, 120(549):535–547.