
DTU

Simon Antoine Savine s204798, December 2024

Phase-Type Distributions: Simulation and Estimation with Machine Learning

Estimation of phase-type (PH) distributions with machine learning

1. Continuous-time Markov chains and phase-type distributions

2. Estimation by machine learning

3. The goal was to combine 1 and 2 and estimate general PH distributions
   Sadly, we didn't get this far
   But we did build a very effective neural network for estimation
   And obtained promising results for special PH distributions

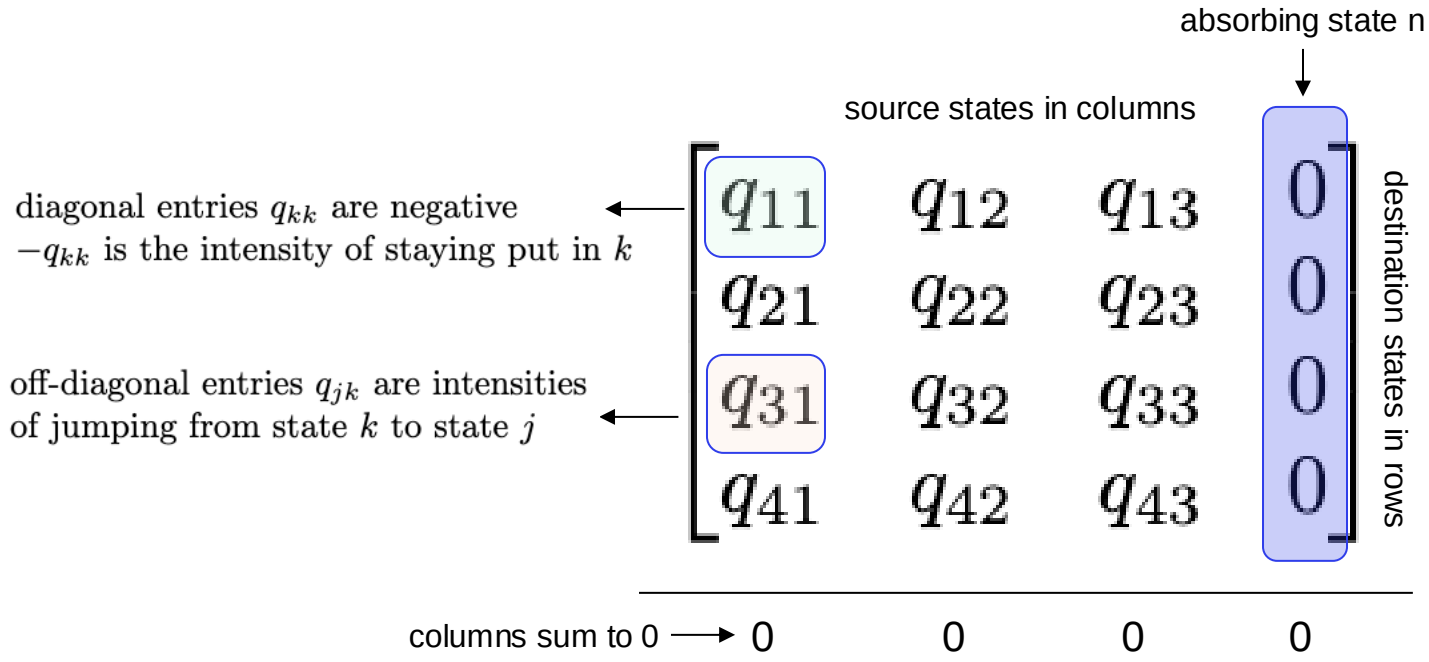
Continuous-time Markov chain (CTMC)

- PH distributions are distributions of absorption time of CTMCs – let us start there
- CTMC models a system that continuously transitions between n states
  - Atom transitioning between quantized excitation states before reaching ground state
  - Company transitioning between credit ratings before going to default
- Parameters:
  - Intensity matrix Q, n by n
  - Initial probabilities P0, size n

Continuous-time Markov chain (CTMC)

P0 = state probabilities at time 0, sum to 1 = $\begin{bmatrix} p_1 & p_2 & p_3 & 0 \end{bmatrix}$

Q =



intuition: "$Q = (R - I)/\Delta_t$"
R: state transition probability matrix
I: identity matrix

→ $P_t = e^{Qt} P_0$

Phase-type (PH) distributions

- Holding time
  - Definition: holding time τ_k: given the system is in state k, exit time from state k
  - Proposition: $\tau_k \sim Exp(-q_{kk})$
- Absorption time
  - Definition: absorption time τ: time when the system first hits absorbing state n
  - Theorem : $F_\tau(t) = Pr(\tau < t) = Pr(X_t = n) = \alpha_n^T P_t = \alpha_n^T e^{Qt} P_0$ and $f_\tau(t) = \frac{dF_\tau(t)}{dt} = \alpha_n^T e^{Qt} Q P_0$

α_n: vector with entries 0 except 1 in position n

- PH distribution
  - Distribution of the associated CTMC's absorption time
  - Same parameters n, P0 and Q as associated CTMC
  - Conversely, pdfs of this form define PH distributions and have an associated CTMC with same parameters

Invalid PH distributions

- Example: $P_0 = \begin{bmatrix} p_1 & p_2 & 0 & 0 \end{bmatrix}^T$ and $Q = \begin{bmatrix} -\lambda_1 & \lambda_2 & 0 & 0 \\ \lambda_1 & -\lambda_2 & 0 & 0 \\ 0 & 0 & -\lambda_3 & 0 \\ 0 & 0 & \lambda_3 & 0 \end{bmatrix}$
  - System starts in state 1 or 2, and stuck forever between the two
  - Never gets absorbed
  - PH CPD and pdf are uniformly 0

- Invalid PH when there is no positive probability path from initial to absorbing state

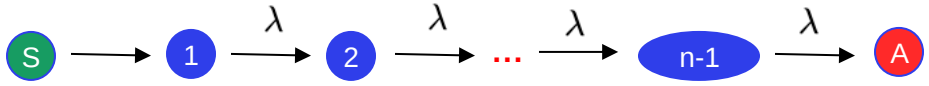
Special PH: exponential distributions

- When n = 2
  - $P_0 = \alpha_1$ and $Q = \begin{bmatrix} -\lambda & 0 \\ \lambda & 0 \end{bmatrix}$

  - Absorption time = holding time
  - Hence, PH = Exp(lambda)
- PH distributions include exponential distributions

Special PH: Erlang distributions

- Another special case with arbitrary n:
  - $P_0 = \alpha_1$ and $Q = \begin{bmatrix} -\lambda & 0 & 0 & 0 \\ \lambda & -\lambda & 0 & 0 \\ 0 & \lambda & -\lambda & 0 \\ 0 & 0 & \lambda & 0 \end{bmatrix}$



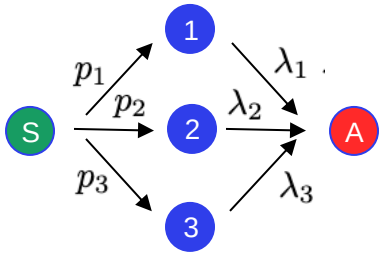
- Absorption time = sum of the n-1 holding times $\tau = \sum_{k=1}^{n-1} \tau_k$

- The distribution of a sum of independent exponentially distributed variables with same rate lambda is: Erlang(n-1, lambda)

- PH distributions include Erlang distributions

Special PH: mixture of exponential distributions

- Another special case:
  - General $P_0 = [p_1, p_2, p_3, 0]^T$ and $Q = \begin{bmatrix} -\lambda_1 & 0 & 0 & 0 \\ 0 & -\lambda_2 & 0 & 0 \\ 0 & 0 & -\lambda_3 & 0 \\ \lambda_1 & \lambda_2 & \lambda_3 & 0 \end{bmatrix}$



- Absorption time:
  - $\tau \sim Exp(\lambda_i)$ with probability p_i
  - This is, by definition, a mixture of exponential distributions
- PH distributions include mixtures of exponential distributions

Special PH: general mixtures

Any mixture of PH is a PH

– Consider the 2 PH $(P_0^{(1)}, Q^{(1)})$ and $(P_0^{(2)}, Q^{(2)})$.

– Build the mixture PH P and Q with probabilities p1 and p2:

P =

$p_1 \widetilde{P_0^{(1)}}$	$p_2 \widetilde{P_0^{(2)}}$	0

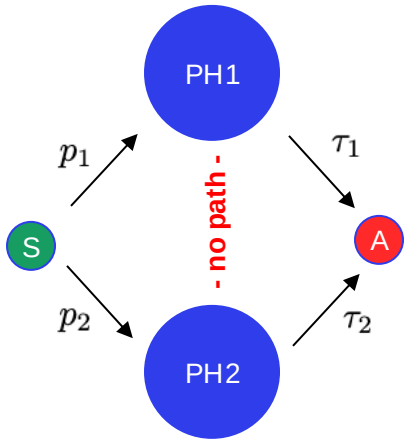
$\widetilde{P_0}$ the probability vector P_0 deprived of its last entry 0.

Q =

$\widetilde{Q^{(1)}}$	0	
0	$\widetilde{Q^{(2)}}$	0
$q_n^{(1)}$	$q_n^{(2)}$	

$\widetilde{Q}$ the intensity matrix deprived of its last column

q_n the last row of the intensity matrix Q.



PH distribution: expressiveness

- PH distributions can represent exponentials, Erlang, or any mixture of other PH including mixtures of Erlang
- It turns out they can represent *any* positive-valued distribution
  (like polynomials represent all smooth functions)

> **Theorem** (Density of PH distributions). *The set of phase-type distributions is dense in the field of all positive-valued distributions, that is, it can be used to approximate any positive-valued distribution to arbitrary precision.*

- In return:
  - Analytic manipulation is nontrivial: matrix exponential in the pdf expression
  - Simulation (via associated CTMC) is computationally heavy
  - Estimation is very hard

Simulation of PH distribution

- Direct simulation is intractable (to best of our knowledge)
- Simulate the associated CTMC:
  - Holding time in state k follows Exp(-qkk)
  - When exiting state k, probabilities of next state are Pj = qjk / -qkk

Algorithm (Simulation of CTMC). *A CTMC with initial probabilities P_0, intensity matrix Q and one absorbing state n is simulated as follows:*

1. *Draw initial state k from distribution P_0. Memorize the resulting state $X_0 = k$ at time $t = 0$.*
2. *Repeat until absorbing state $k = n$:*
   (a) *Draw the holding time τ_k from exponential distribution with rate $\lambda = -q_{kk}$. Update current time $t = t + \tau_k$.*
   (b) *Draw the next state j from the distribution $P_j = q_{jk}/-q_{kk}$. Update current state to $k = j$ at time t.*

Simulation of PH distribution

- Simulate bimodal mixture of Erlang(2, 0.5) and (15, 1) with probabilities 0.5 and 0.5
- PH representation:

```
P0
[0.5 0.  0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
Q
[[-0.5  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.5 -0.5  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.  -1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   1.  -1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   1.  -1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   1.  -1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   1.  -1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   1.  -1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   1.  -1.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   1.  -1.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   1.  -1.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.  -1.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.  -1.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.  -1.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.  -1.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.  -1.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.  -1.   0. ]
 [ 0.   0.5  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0. ]]
```

Simulation of PH distribution

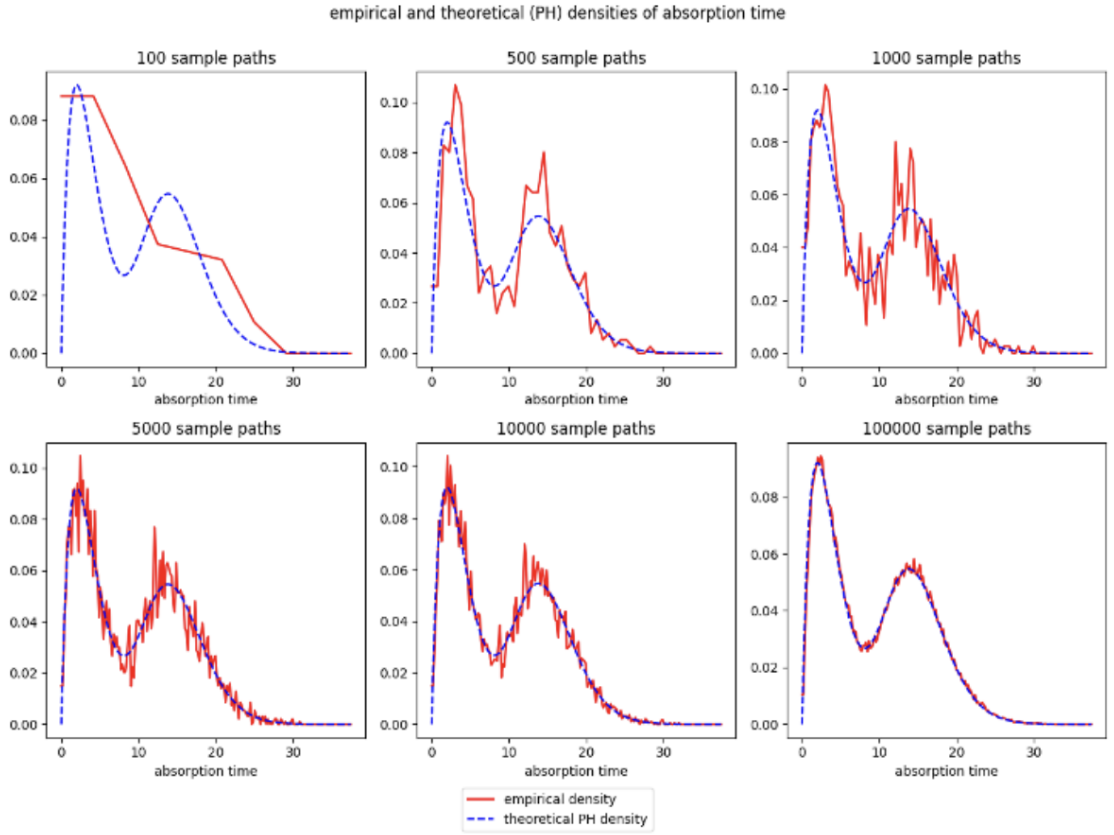


Figure 4.5: Repeat the experience of figure 4.3 with bimodal 50/50 mixture of Erlang(2, 0.5) and Erlang(15, 1), represented as a PH with the parameters of figure 4.4


DTU

Statistical estimation by Machine Learning

- Goal: build an **end-to-end** ML model

- Proposed in the paper:

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

- Benefits:
  - A unified machinery for estimating arbitrary distributions, where distributions are plug-and-play (we did not completely achieve this goal but I am now convinced it can be done)
  - A simple alternative when the derivation of MLE or algorithm such as EM is very hard (such as a PH)
  - A working solution when MLE is intractable (such as multivariate PH)

Natural methodology

1. Simulate the dataset
   A. For each training example, pick a set of parameters
   B. Simulate a sample with these parameters
   C. Reverse the relationship: the sample is the training input, the parameters are the training labels

2. Train a neural network (ANN) on this dataset, by minimization of MSE

3. Evaluate the ANN on unseen data

4. The trained ANN is ready to make estimations

- Sounds easy:
  - All we need is a distribution with a sampling routine
  - We can simulate while training → no limitation of data → can't overfit
  - ANN machinery (building blocks and layer types, gradient descent, backpropagation) available in public frameworks like Google's TensorFlow

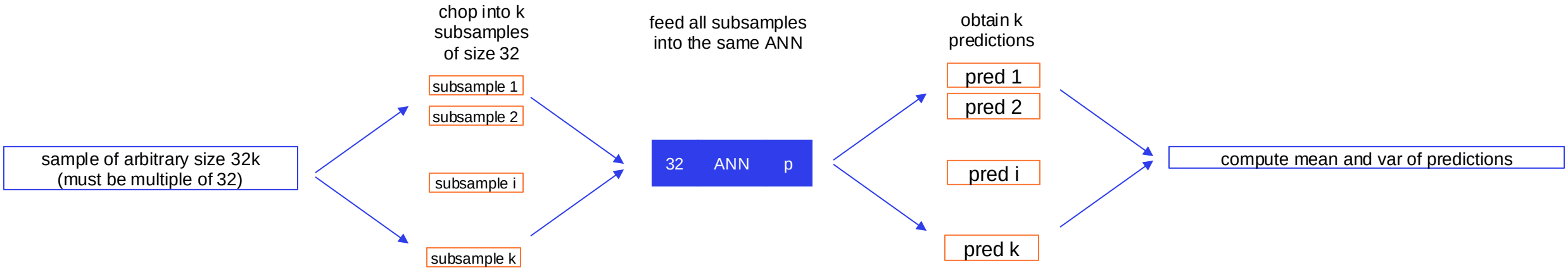
Challenges

- Labels are not a deterministic function of inputs
  - Learning a deterministic formula such as $X \to \frac{1}{\bar{X}}$ (MLE for exponential rate) is very easy
  - But useless: if you already have a formula, you don't need estimation by ML
  - Here we learn to predict generative parameters from noisy samples – much much more difficult
- Yet, it was easy to produce a proof of concept with decent results on fixed-size samples
  The harder challenges were:

1. **Estimators should accept samples of arbitrary size and behave as expected, so that its variance decreases with sample size, without increasing bias**
2. Estimation should be permutation-invariant: it is not acceptable to obtain a different estimate by shuffling the sample
3. It is hard to obtain unbiased estimators that compete with MLE – good progress but scope for further research

First attempt

- Our first attempt at addressing those challenges was:
  - Train an ANN with samples of small size (32)
  - For estimation:
    - Split the sample into k samples of size 32
    - Obtain k predictions
    - Produce the average prediction as the estimate



First attempt

- (Apparently) addressed challenges correctly:
  - Accepts samples of ‘almost’ arbitrary size 32k (or drops the reminder)
  - Variance of estimator proportional to 1 / k (avg of independent variables) , hence to 1 / n (n: sample size)
  - Not permutation-invariant, but this is easily resolved:
    - The ANN learns by itself an ‘almost’ permutation-invariant function from the examples in the dataset This is just a tad wasteful
    - Or we could sort the small samples in increasing order so the ANN sees them in a ‘canonical’ form
  - (With some tinkering about the ANN architecture and training loop) we obtained decent results, roughly on par with MLE for simple distributions (exponential, normal)
  - Additional benefit: provides a standard error estimate (sqrt(variance / k)) along with estimation

First attempt

- It turns out that this methodology:
  1. Is fundamentally incorrect
  2. Necessarily results in bias

- To see this, consider the exponential distribution

- The MLE estimate is $\widehat{\lambda} = \frac{1}{\overline{X}}$
- Suppose the ANN learns the MLE so predictions from the k small samples are $\widehat{\lambda}_k = \frac{1}{\overline{X_k}}$
- The overall prediction is the mean of the k predictions: $\widehat{\lambda} = \overline{\widehat{\lambda}_k} = \overline{\frac{1}{\overline{X_k}}}$
- Which is not the correct estimation $\lambda* = \frac{1}{\overline{X}} = \frac{1}{\overline{X_k}}$
- By Jensen's inequality (since the inverse function is convex) $\widehat{\lambda} > \lambda*$
- Hence, the methodology produces a bias by construction

Second attempt

- We found a correct methodology
  - Unfortunately, (very) late in the project
  - Inspired by the Deep Sets paper, famous with the ML community

[9] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola. Deep Sets. *arXiv e-prints*, page arXiv:1703.06114, Mar. 2017.

- This really turned things around
- And provides ideas for further research

Deep Sets : Theory

- Fundamental theorem

Theorem (Deep Sets). *Any permutation-invariant function g from $\mathbb{R}^n$ to $\mathbb{R}^p$ can be written as:*

$$g(x) = h\left[\frac{1}{n}\sum_{i=1}^{n} e(x_i)\right]$$

where:

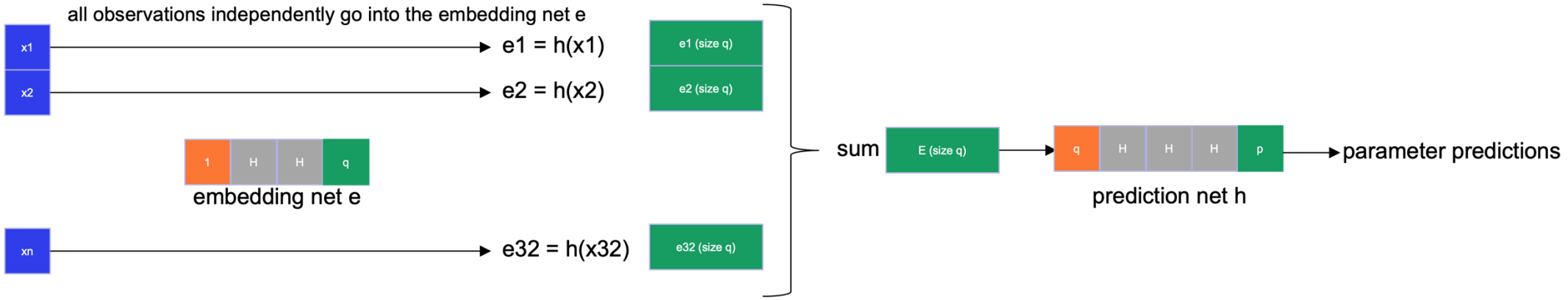
- *e is a* scalar *function, called* embedding function, *from $\mathbb{R}$ into some embedding space E. In theory, the embedding space may be of infinite dimension. In practice, we work with $E = \mathbb{R}^q$, where q is the embedding size. It helps to think of e as a collection of q scalar functions e_j from $\mathbb{R}$ to $\mathbb{R}$. The same q embeddings are applied to the n observations x_i.*
- *the q embeddings of the n observations x_i are averaged across observations, so that the aggregated embedding $\overline{e(xi)}$ is one vector of size q.*
- *the prediction function h from E (in practice: $\mathbb{R}^q$) to $\mathbb{R}^p$ makes a prediction only based on the aggregated embedding.*

Deep Sets : Examples

- Easy to see that this construction is permutation-invariant
- Harder to accept that all permutation-invariant functions are of this form
- A few examples help:
  - Product function: $g(x) = \prod_{i=1}^{n} x_i : e(x) = \log x \text{ and } h(y) = e^{ny}$
  - Maximum function: $g(x) = \max_{i \in [1,n]}\{x_i\} : e(x) = e^{\alpha x} \text{ and } h(y) = \log(ny)/\alpha$ (when α grows to infinity)
  - MLE for exponential rate: $g(x) = 1/\overline{x} : e(x) = x \text{ and } h(y) = 1/y$
  - MLE for normal mean and variance $g(x) = (\overline{x}, \overline{x^2} - \overline{x}^2) : e(x) = (x, x^2) \text{ and } h(y_1, y_2) = (y_1, y_2 - y_1^2)$

Deep Sets : Architecture

- Represent the 2 functions: embedding and prediction, by ANNs
- Architecture follows



Deep sets

- Developed for permutation-invariance
- Also resolves the (much harder) problem of correctness with respect to sample size
  - Let us revisit the exponential rate:
    - The MLE estimate is $\widehat{\lambda} = \frac{1}{\overline{X}}$, its representation is $e(x) = x$ and $h(y) = 1/y$
    - Suppose the ANN, trained with samples of some size, learns these embedding and prediction functions
    - Then it correctly predicts $g(x) = h[\overline{e(x)}] = 1/\overline{x}$ irrespective of size -- no more bias!
    - Because it learned a *formula* that is size-agnostic (safe for averaging)
    - And it can even correctly predict rates of magnitude unseen in training (since it learned a formula)
  - And the averaging of embeddings mechanically reduces standard error for larger samples

Overcoming bias

- The Deep Set resolves size and permutation invariance
- Remains the problem of bias
- The solution we implemented:
  - During training, feed not 1 but k samples through the Deep Set, all sampled with the same parameters
  - Obtain k predictions for the k samples
    Estimate bias = average prediction – true parameters, and variance (of the k predictions)
  - Minimize loss = a bias^2 + (1 – a) variance – if a=0.5 this is the MSE but we set higher, generally 0.9 to 0.99

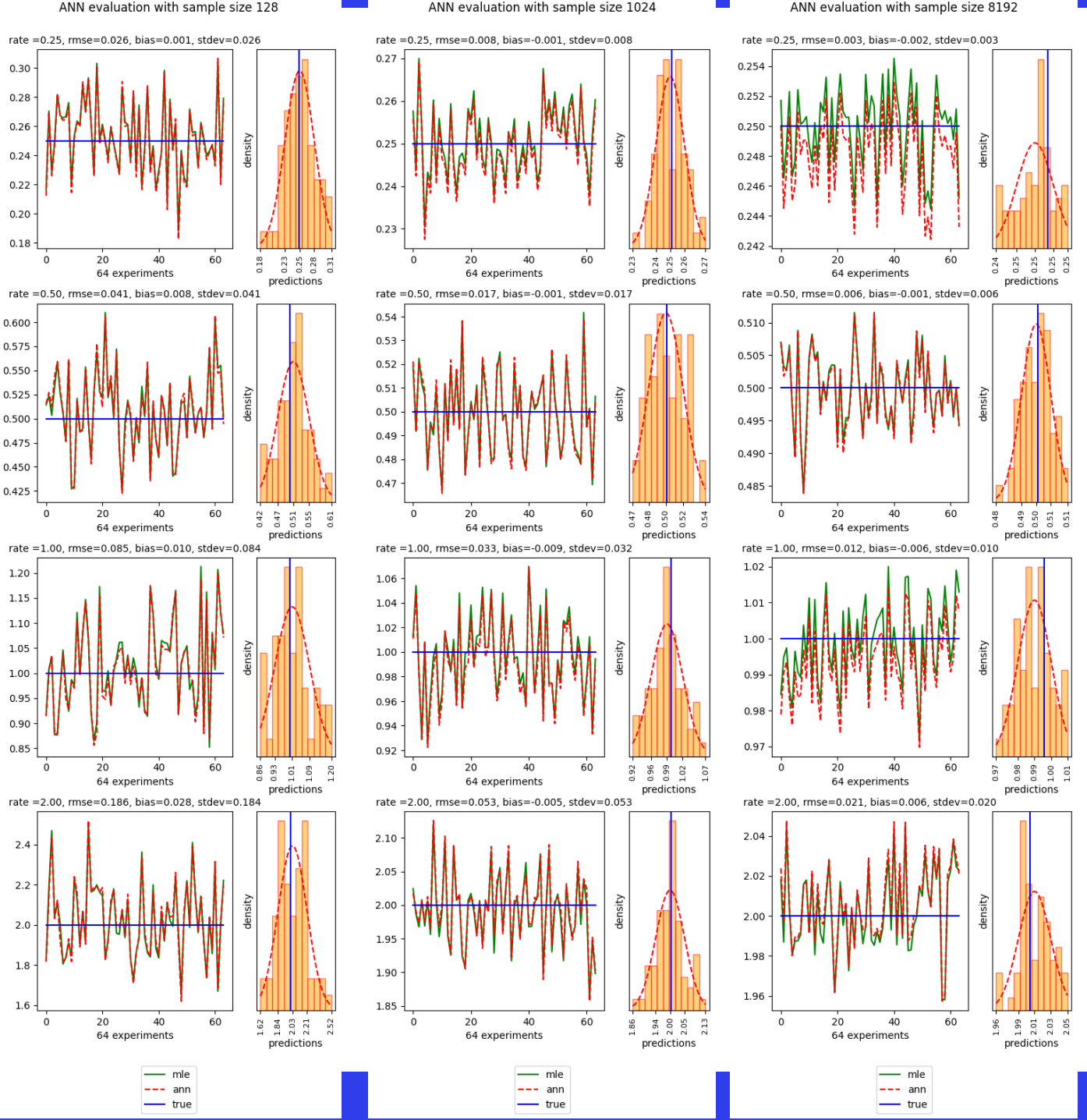
Overcoming bias

- This machinery helps (a lot) but not a silver bullet
- We had to tinker with the training loop, distribution by distribution
  - Bias weight a in loss function
  - Number of epochs and number of batches per epoch
  - Number of examples in every batch for the gradient descent step
  - Learning rate schedule
  - Number of samples and sample size during training
- We set those by “manual gradient descent”, differently for every distribution
- Topic for further research: find automated heuristics
- Another topic for research: investigate training with multiple sizes

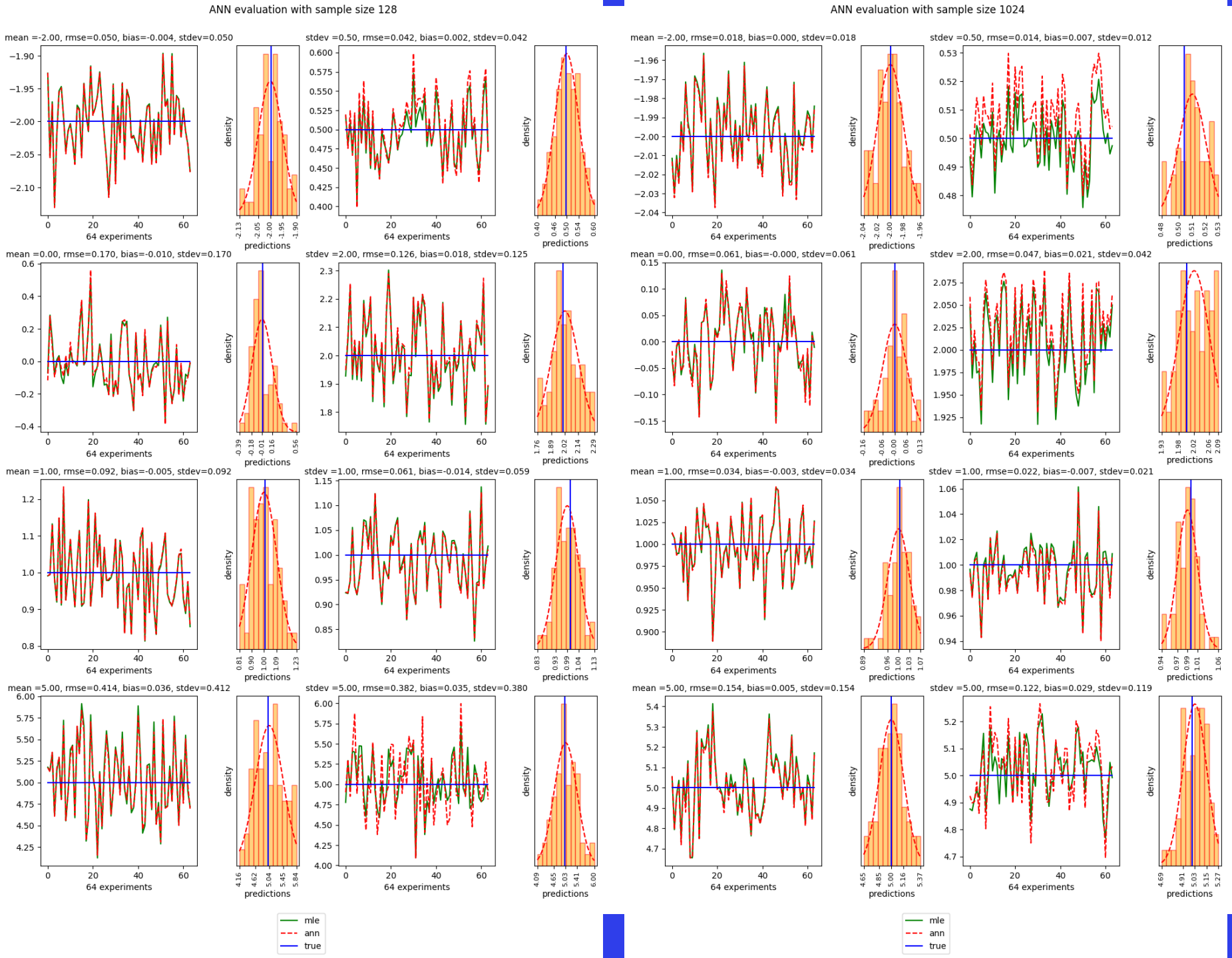
Performance evaluation

- Pick some parameters (manually)
- Generate 64 samples of various sizes (128, 1024, 8192), obtain estimates, compare with truth
- Feed the same samples to MLE estimates for comparison
  - Gamma: use scipy.gamma.fit() – estimates MLE, not clear how from documentation
  - Mixtures: use EM implementation by Jacob Schreiber (Stanford) in open-source library pomegranate
    (I didn't have time to learn, let alone implement EM)

Results: exponential



Results: normal

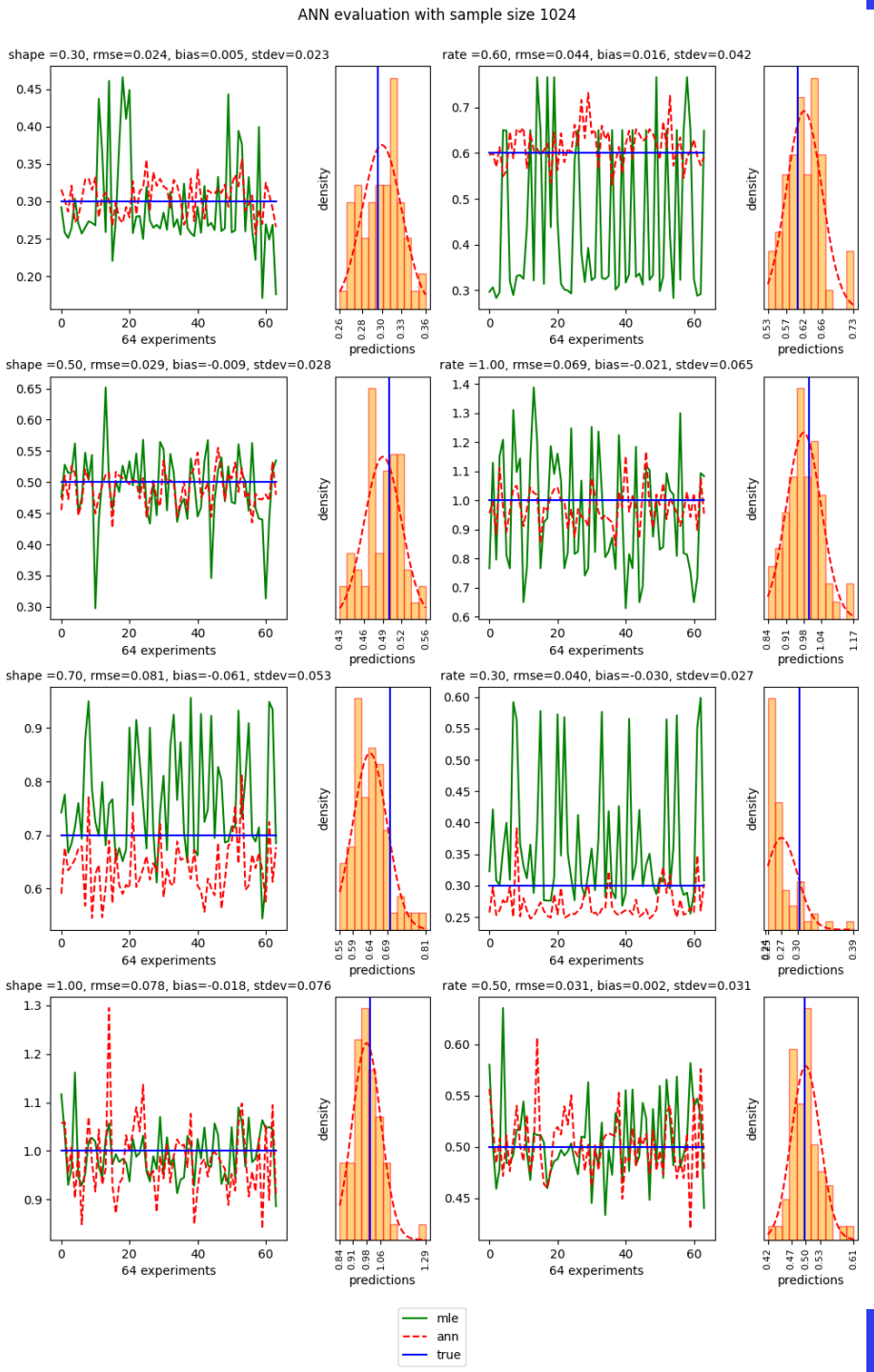


Results: gamma/Erlang

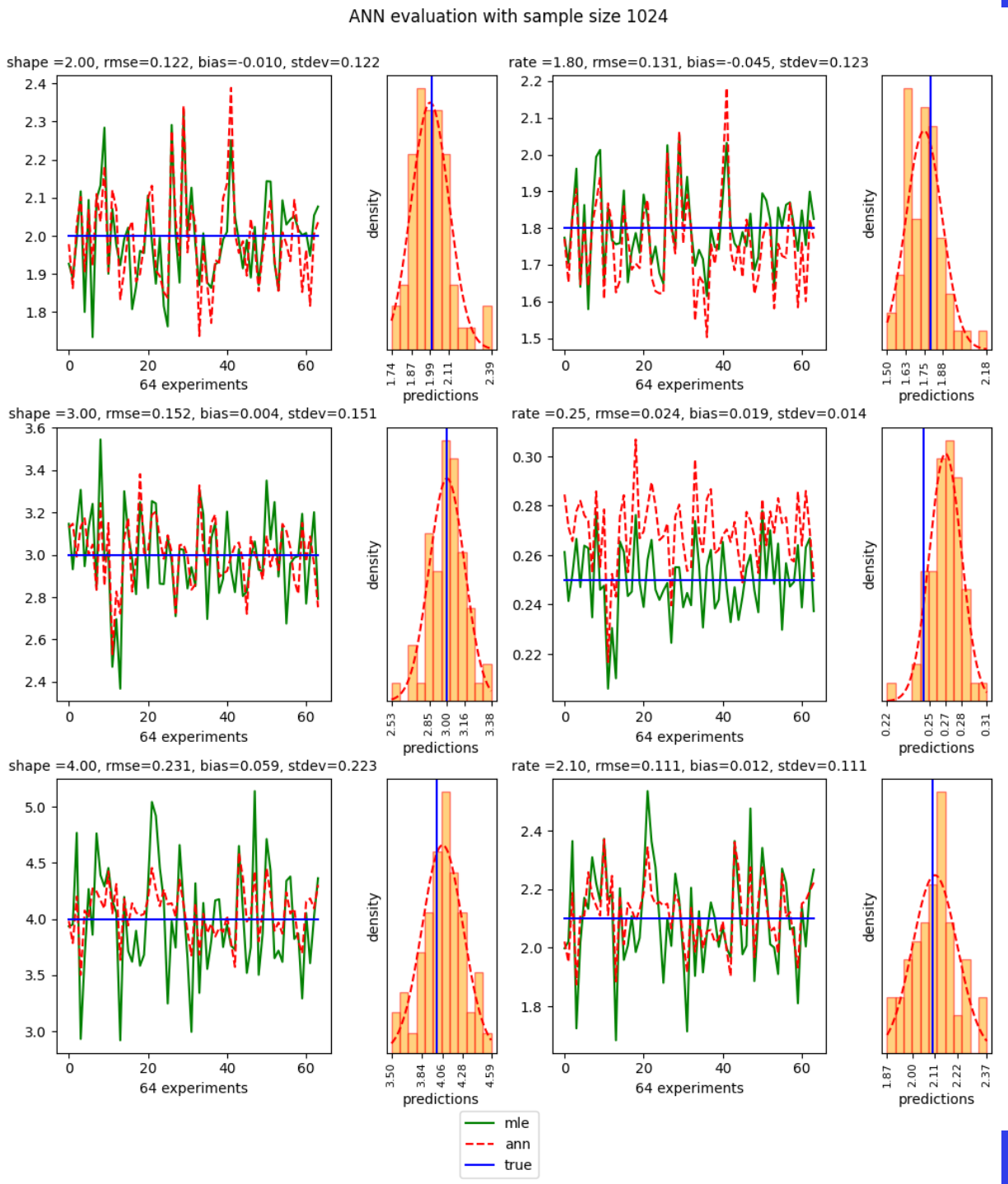
- We only did gamma
  - Can predict Erlang, just crop shape prediction to nearest integer
  - It was suggested to force the ANN to return integers instead, one topic for further research

- For some reason, ANN struggles with small rates (<0.25)
  - This matter needs investigating
  - For now, moved all training rates (and shapes) up by 0.25 from the training set, to prevent interference
  - Explains abysmal performance with low rate configurations
  - Contrarily to exponential or normal, does not extrapolate well to out-of-distribution configurations

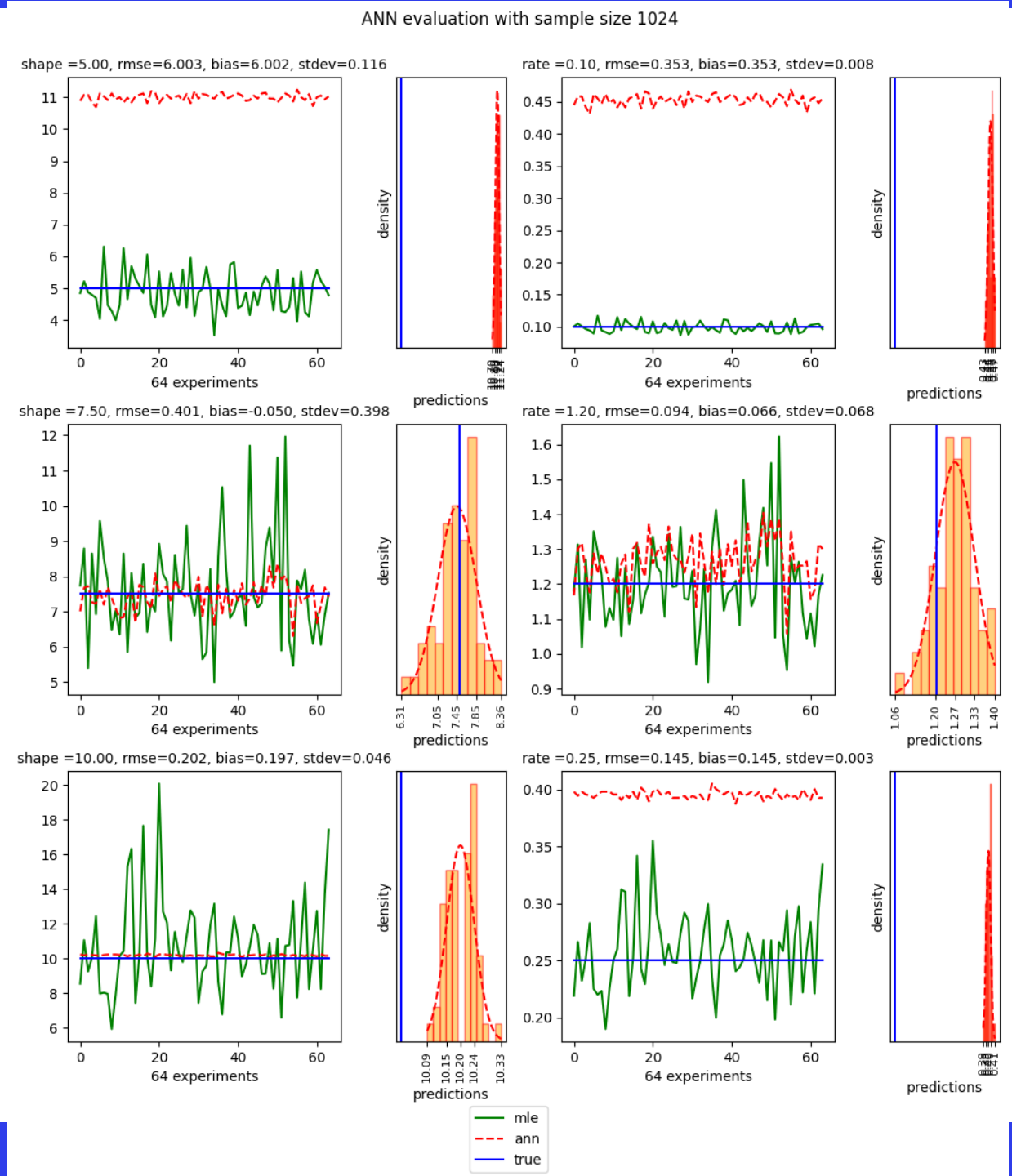
Results: gamma/Erlang – small shape



Results: gamma/Erlang – medium shape



Results: gamma/Erlang – large shape



Results: mixture of 2 exponential distributions

- Mixtures are subject to non-unique representations

1. (p1, p2, lam1, lam2) is the same as (p2, p1, lam2, lam1)
   -> resolved by sorting parameters in increasing order of rates

2. - when a probability is near 0, the rate cannot be predicted
   - when the two rates are close, probabilities cannot be predicted
   -> resolved by under-weighting examples with similar rates or low minimum probability

- Only then could the ANN learn

Results: mix of 2 exp

Topics for further research

- Further fine-tune the ANN and the training loop. For instance, explore the idea of training on samples of different sizes.
- Develop a truly distribution-independent software, so that users can plug and play with arbitrary distributions. Research training loop heuristics that work well for all distributions without manual tinkering.
- Investigate and improve estimation for gamma and Erlang distributions. Experiment with multiple ANNs depending on the magnitude of the shape parameter.
- Test on many more distributions, particularly, mixtures of more than 2 distributions, up to and including general PH distributions.
- Discriminate between distributions, not only predict parameters. For a given sample, return the most likely distribution, maybe based on likelihood, along with parameter estimates.


DTU