Survival Analysis Using a Bayesian Neural Network

Radford M. Neal

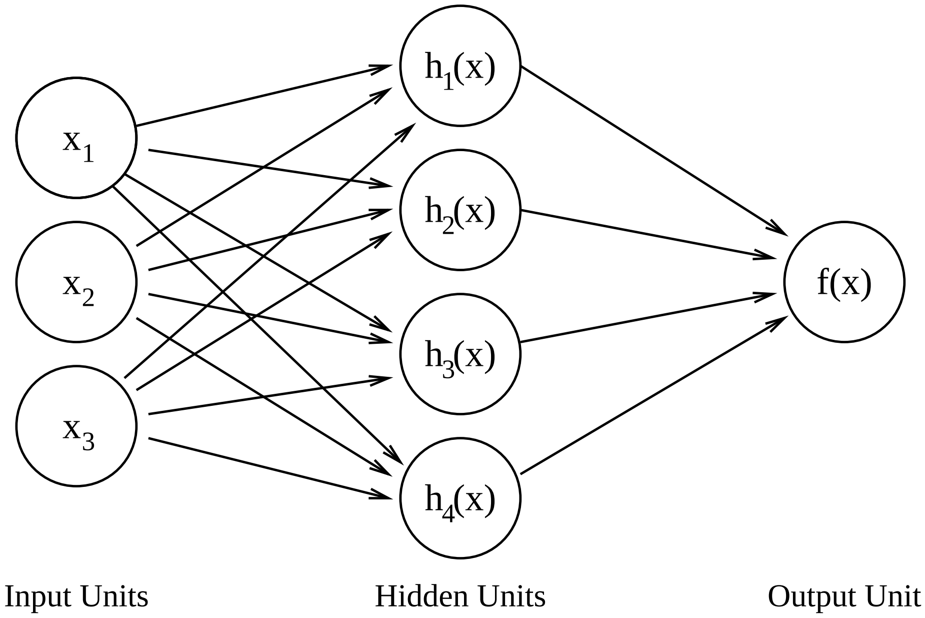
Dept. of Statistics and Dept. of Computer Science
University of Toronto

Joint Statistical Meetings, August 2001

Outline

- Using neural networks to build flexible models.
  - Multilayer perceptron ('backprop') neural networks.
  - Bayesian neural network learning.
- Survival analysis using a Bayesian neural network.
  - Network architectures that define proportional and non-proportional hazard models.
  - Graphical methods for making sense of the results.
- Example: Data on primary biliary cirrhosis (PBC) analysed by Fleming & Harrington.

A Multilayer Perceptron Network



The network takes inputs $x_1, \ldots, x_I$ and from them computes an output, $f(x)$, using a layer of H hidden units:

$$
\begin{aligned}
f(x) &= b + \sum_{j=1}^{H} v_j h_j(x) \\
h_j(x) &= \tanh\left(a_j + \sum_{i=1}^{I} u_{ij} x_i\right)
\end{aligned}
$$

Typically, the function $f(x)$ is used to define the conditional distribution for a response, y, for covariates x — eg, Gaussian with mean $f(x)$.

Conventional Neural Network Learning

Conventional neural network learning can be viewed as *maximum likelihood* estimation for the network parameters — i.e. we find values for the weights and biases, $\hat{w}$, to maximize

$$\prod_{c=1}^{n} P(y^{(c)}\,|\,x^{(c)},\,w)$$

where $(x^{(c)}, y^{(c)})$ are the covariates and response for training case c.

We then make predictions for a test case with covariates $x^{(n+1)}$ using the conditional distribution $P(y^{(n+1)}\,|\,x^{(n+1)},\,\hat{w})$.

Maximum likelihood is prone to "overfitting" when the number of network parameters is large in relation to the number of training cases.

Bayesian Neural Network Learning

Bayesian predictions are found by *integration* rather than maximization. For a test case with covariates $x^{(n+1)}$, we predict $y^{(n+1)}$ using

$$P(y^{(n+1)} \mid x^{(n+1)}, (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}))$$

$$= \int [dw] \, P(y^{(n+1)} \,|\, x^{(n+1)}, w)$$

$$\times \; P(w \,|\, (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}))$$

The posterior distribution used above is

$$P(w \,|\, (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}))$$

$$\propto \; P(w) \; \prod_{c=1}^{n} P(y^{(c)} \,|\, x^{(c)}, w)$$

We must have a *prior distribution* for the weights, $P(w)$. For example, this might be Gaussian, with the weights being independent, but with weights of different types having different variances.

Complexity in Bayesian Models

The model and prior in Bayesian learning represent beliefs about the problem based on prior knowledge. If we believe the problem is complex, we should use a complex model.

Overfitting should *not* be a problem with Bayesian learning. It is usually best to use a network with a large number of hidden units (limited by computational cost).

For a complex problem, the appropriate prior is often most conveniently expressed using *hyperparameters*, which control the priors for lower-level parameters.

In such a *hierarchical model*, the prior for the low-level parameters (weights & biases), w, is expressed using hyperparameters α as follows:

$$P(w) \;=\; \int [d\alpha]\, P(w\,|\,\alpha)\, P(\alpha)$$

Roles for Hyperparameters

In a simple hierarchical model for a network with one hidden layer, three hyperparameters might be used, controlling the variance of Gaussian priors for three groups of parameters:

- input-to-hidden weights,
- hidden unit biases,
- hidden-to-output weights.

Some more interesting things to do with hyperparameters:

- Use separate hyperparameters for each input. Some inputs (covariates) may then come to have more influence than others.
- Use several hidden layers that look at different subsets of the inputs. Let hyperparameters control how much each layer (additive component) contributes to the function.

Survival Analysis and Hazard Functions

I will use models for survival data based on the hazard function, $h(t,x)$, defined by

$$h(t,x)\,dt \;=\; \text{Pr}\big(\text{person with covariates } x \text{ dies in the interval } (t, t+dt) \;\big|\; \text{they live to time } t\big)$$

Survival probabilities can then be written as

$$\text{Pr}(\text{person with covariates } x \text{ lives to time } t) \;=\; \exp\left(-\int_0^t h(s,x)\,ds\right)$$

The likelihood factor for person i with covariates x_i, known to have died at time t_i, is

$$\exp\left(-\int_0^{t_i} h(s,x_i)\,ds\right) h(t_i,x_i)$$

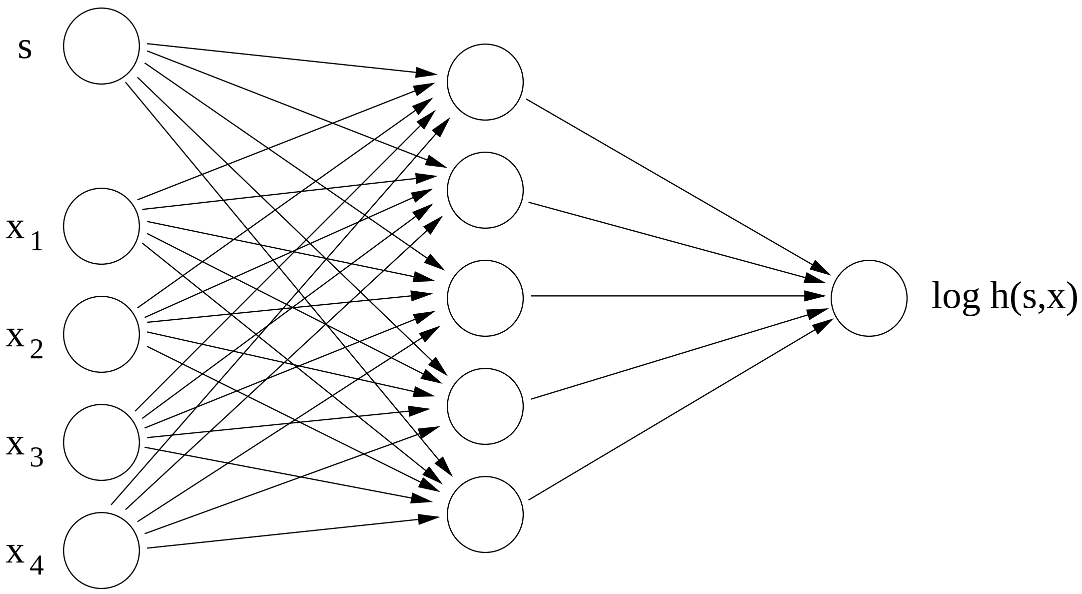
The likelihood factor when person i is known only to have survived to time t_i is

$$\exp\left(-\int_0^{t_i} h(s,x_i)\,ds\right)$$

This assumes that censoring can be regarded as uninformative (eg, occurring at random).

Modeling the Hazard with a Neural Net

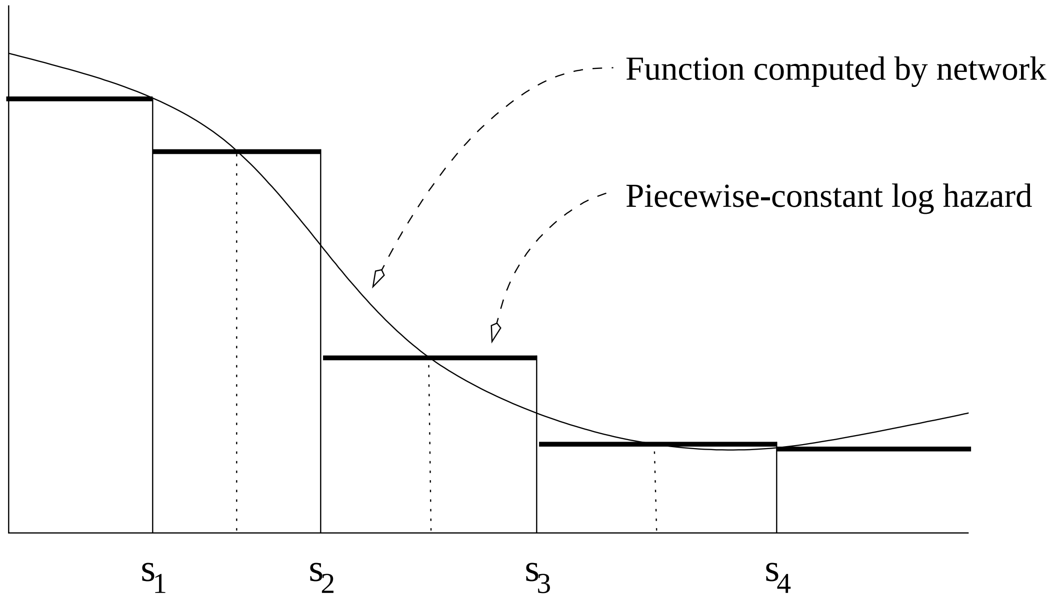
I will use a neural network to model the log of the hazard function, with time and covariates as inputs. For example:



Unfortunately, to use this model directly we would need to compute $\int_0^t h(s,x)\,ds$, in order to evaluate the likelihood. This would require costly numerical integration.

Defining a Piecewise Constant Hazard Using the Neural Network Model

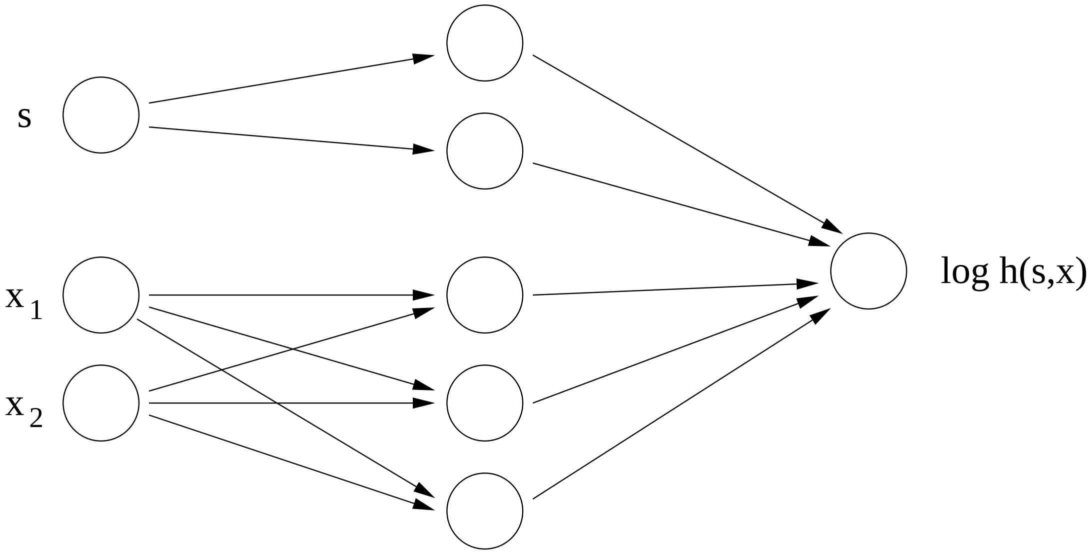
To avoid difficult integrations, I use the neural net to define a model with piecewise-constant hazard. For fixed covariates, x, the log hazard is modelled as follows:



Times $s_1, \ldots, s_k$ are chosen to be dense enough that the breaks don't affect the result much. Using many pieces causes no statistical problems, it just slows the computations.

Networks for Proportional Hazards

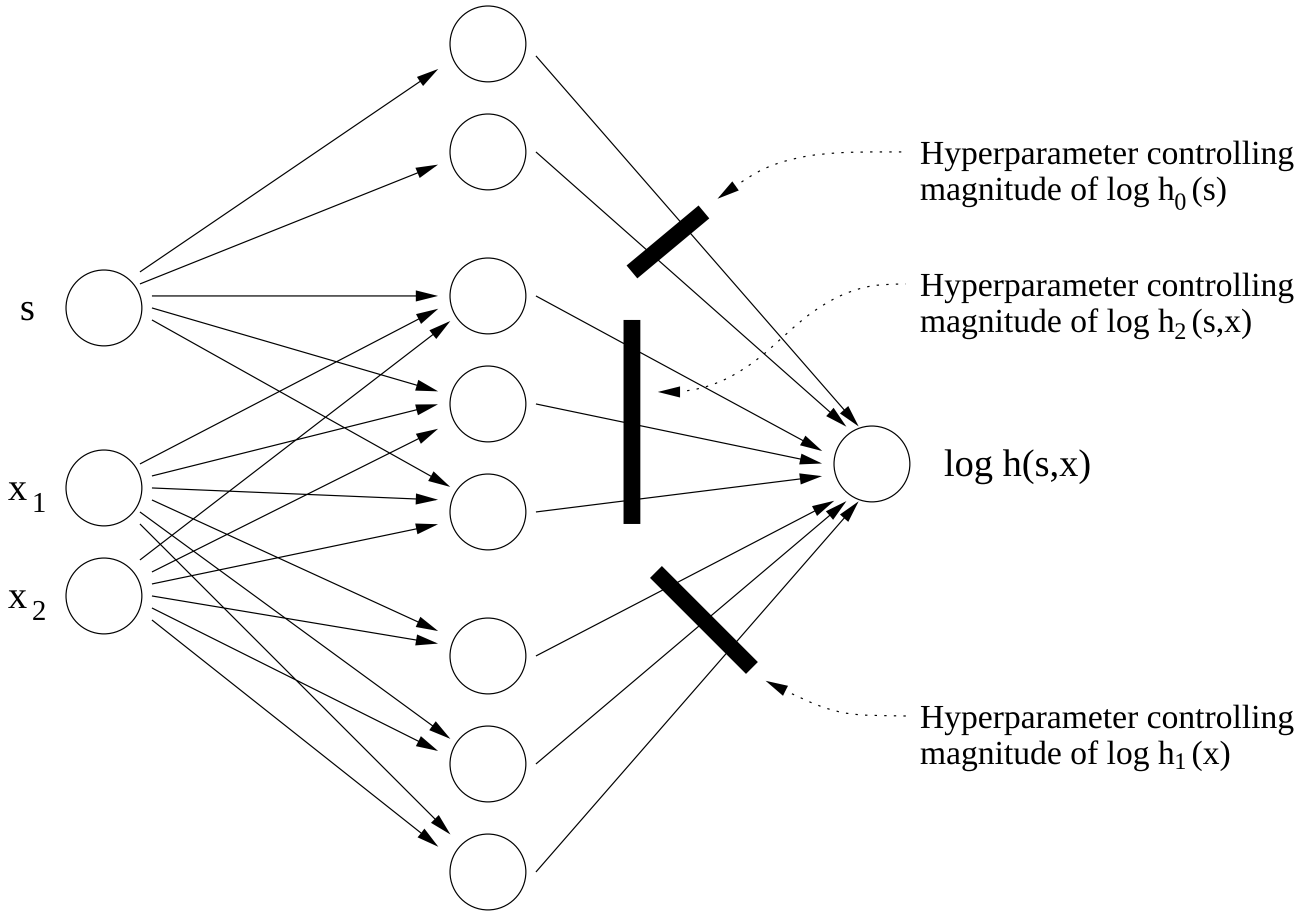
A "proportional hazard" model, in which $\log h(s,x) = \log h_0(s) + \log h_1(x)$ can built using a network like the following:



The upper "layer" of two hidden units computes $\log h_0(s)$; the lower layer of three hidden units computes $\log h_1(x)$.

A Network for Discovering Whether Proportional Hazards are Appropriate

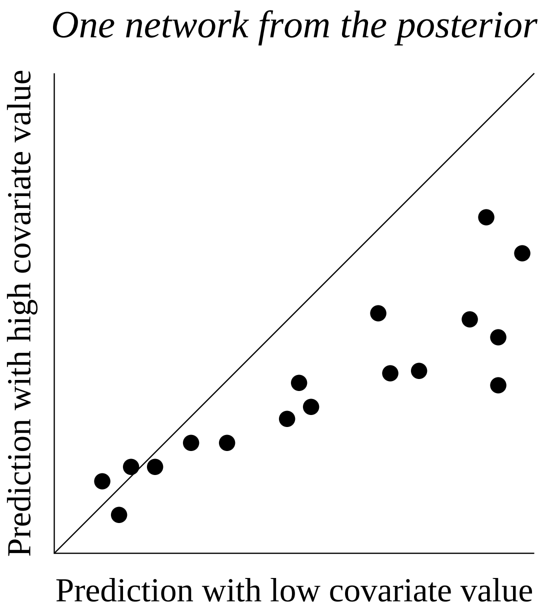
A hierarchical model can "discover" whether a proportional hazards model is appropriate, by using separate hyperparameters to control the magnitudes of additive components:

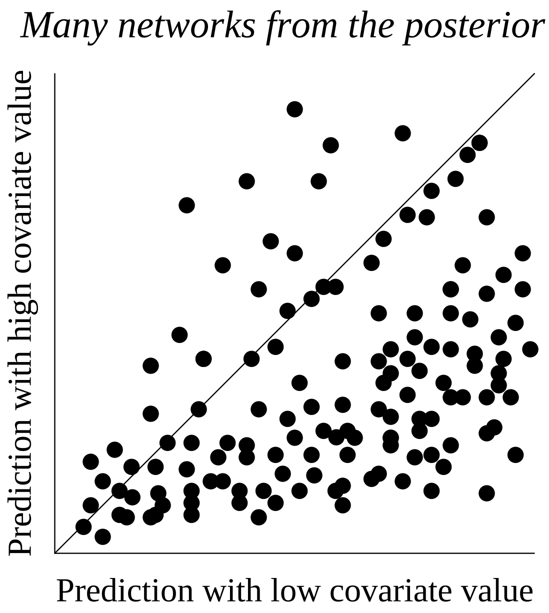


$\log h(s,x) = \log h_0(s) + \log h_1(x) + \log h_2(s,x)$,
but $\log h_2(s,x)$ can disappear if the data say so.

Graphical Display of Covariate Effects

One way to make sense of these complex models is to see how predictions for median survival time change when a covariate changes:





Each dot is a training case. The vertical coordinate is the prediction with a high value for the covariate, with other covariates at their actual values. The horizontal coordinate is the prediction for a low value.

I use 0/1 or actual $\pm$ SD/2 for low and high covariate values.

The PBC Data

Fleming and Harrington use this data as an example in their book. It is from a clinical trial with 312 subjects, testing a drug for treating primary biliary cirrhosis. I looked only at the covariates that F&H mostly looked at:

Covariate	Coding	Transformation
1. drug	0=placebo 1=drug	
2. age	in years	(age-50)/10
3. sex	0=male 1=female	
4. ascites	0=no 1=yes	
5. hepatom	0=no 1=yes	
6. spiders	0=no 1=yes	
7. edema	0=no 0.5=sort of 1=yes	
8. bili	bilirubin in mg/dl	log(bili)
9. albumin	albumin in gm/dl	log(albumin)-1
10. alkphos	alkaline phosphatase in U/liter	log(alkphos)-7
11. platelet	platelets per cubic ml/1000	log(platelet)-5
12. protime	prothrombin time in seconds	log(protime)-2

The transformations were chosen so that, *a priori*, a difference of one is expected to perhaps be associated with a fairly large difference in survival.

Testing Models with Split Data

I randomly split the data into 212 training cases and 100 test cases, to see how complex models compare with simple models.

model /prior	Components present $\log h_0(s)$	$\log h_1(x)$	$\log h_2(s,x)$	Ave. log lklhd train	test
`nlx/1`	5 h.u.	linear	—	-1.073	-1.259
`nlx/2`	5 h.u.	linear	—	-1.063	-1.250
`nnx`	5 h.u.	8 h.u.	—	-1.038	-1.248
`xxn/1`	—	—	10 h.u.	-1.052	-1.247
`xxn/2`	—	—	10 h.u.	-1.034	-1.238
`nln`	5 h.u.	linear	10 h.u.	-1.058	-1.241
`nnn`	5 h.u.	8 h.u.	10 h.u.	-1.040	-1.229

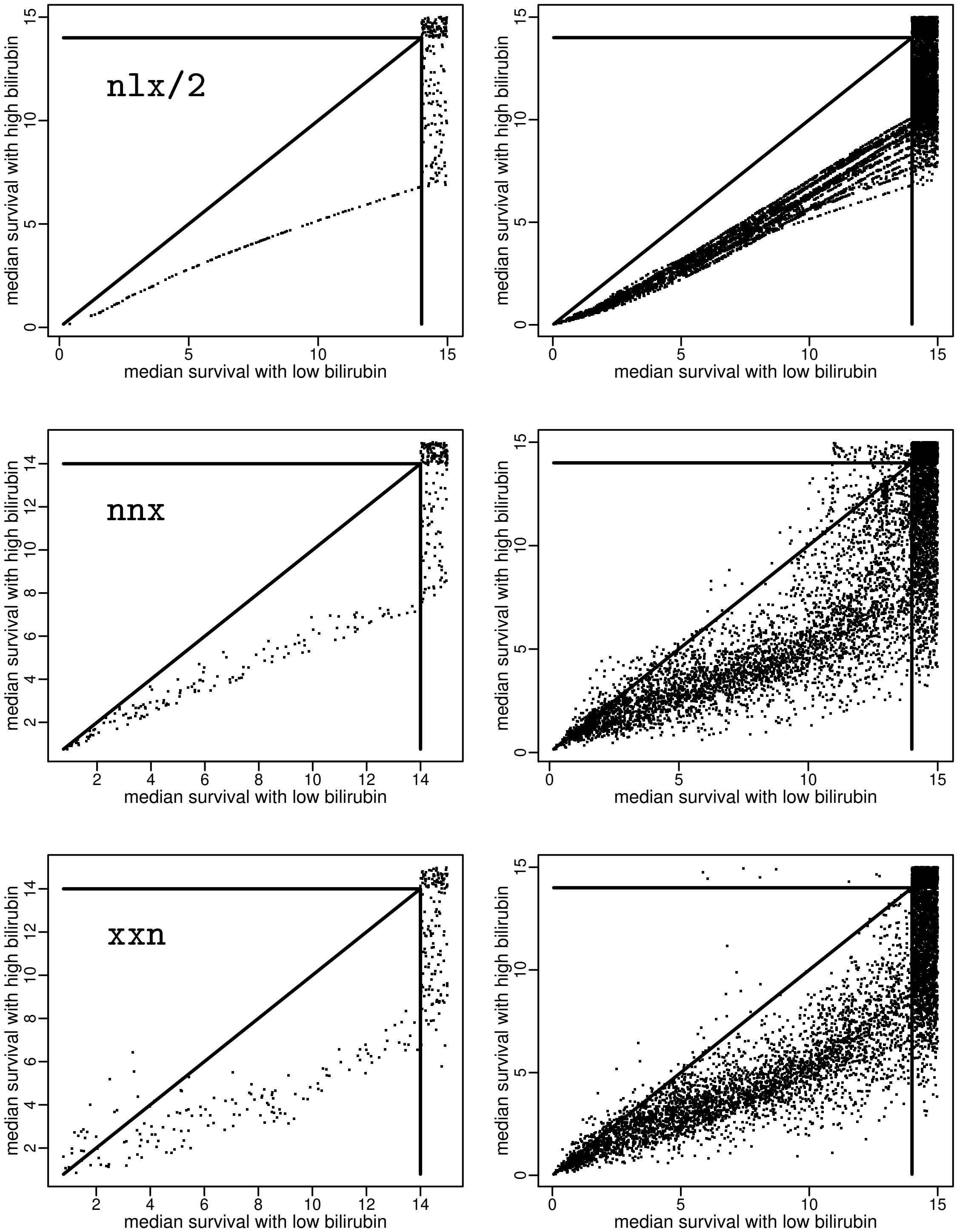
`nlx/2` has separate hyperparameters for each input.
`xxn/2` has its prior for weight variances shifted up by a factor of two.

The more complex models seem to do better on test cases, but none of the differences are statistically significant using a paired t test.

The `nln` model gives about equal posterior probability to almost-proportional and non-proportional hazards. The posterior for the `nnn` model favours non-proportionality.

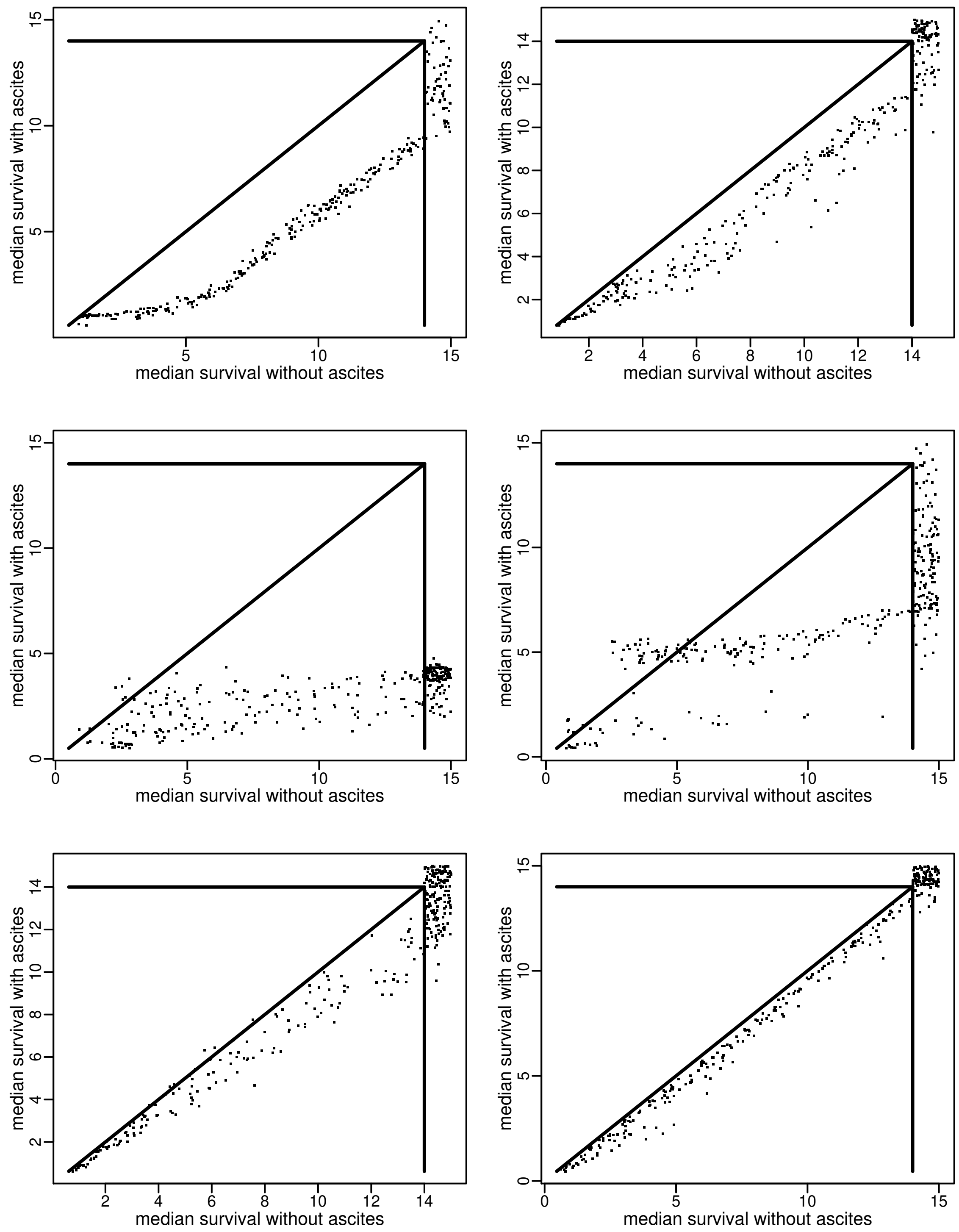
Proportional vs. Non-Proportional Hazards Using All the Data

One network from posterior   25 networks from posterior



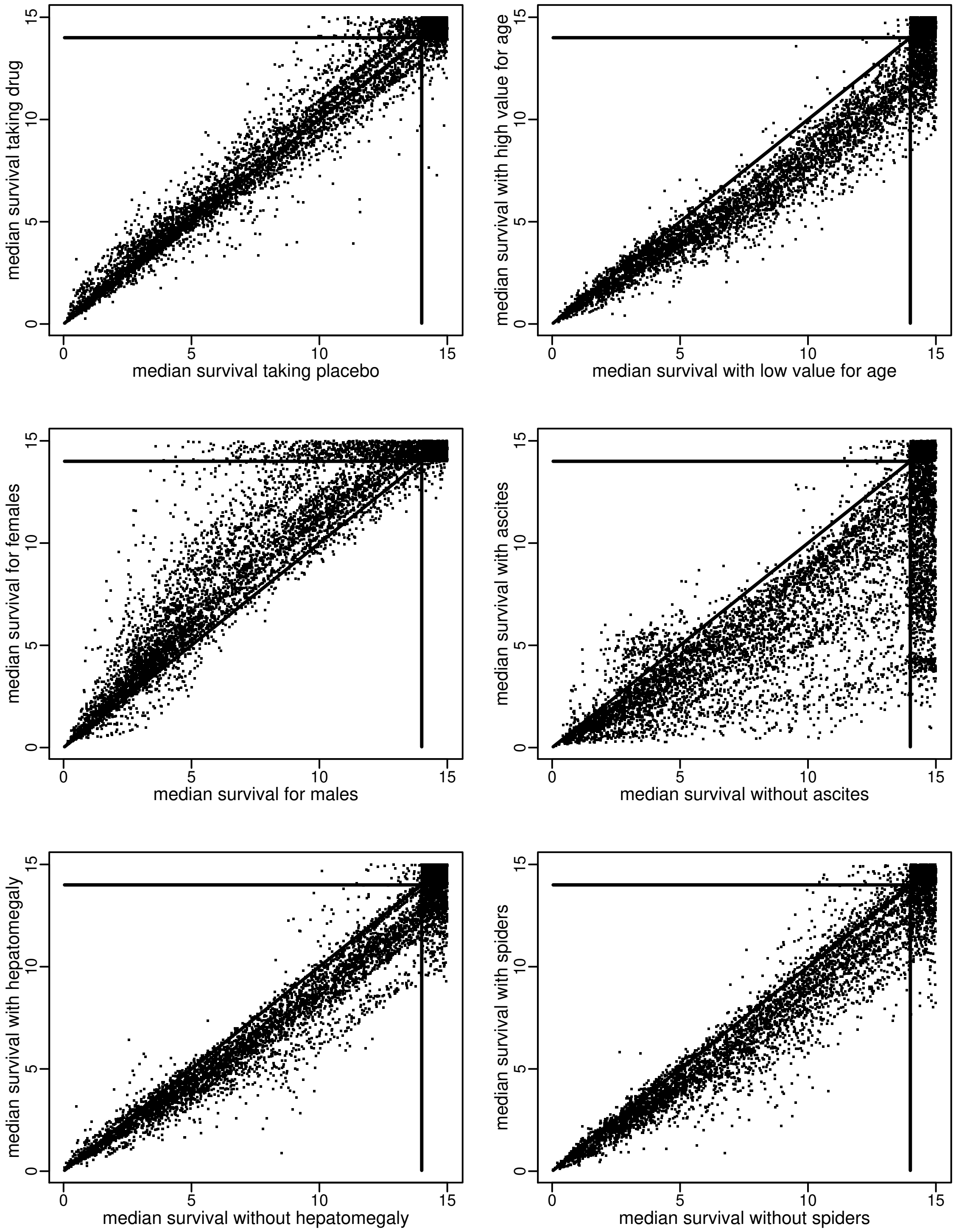
Posterior Distribution of an Effect

Using 6 networks from the posterior of model `nnn`.



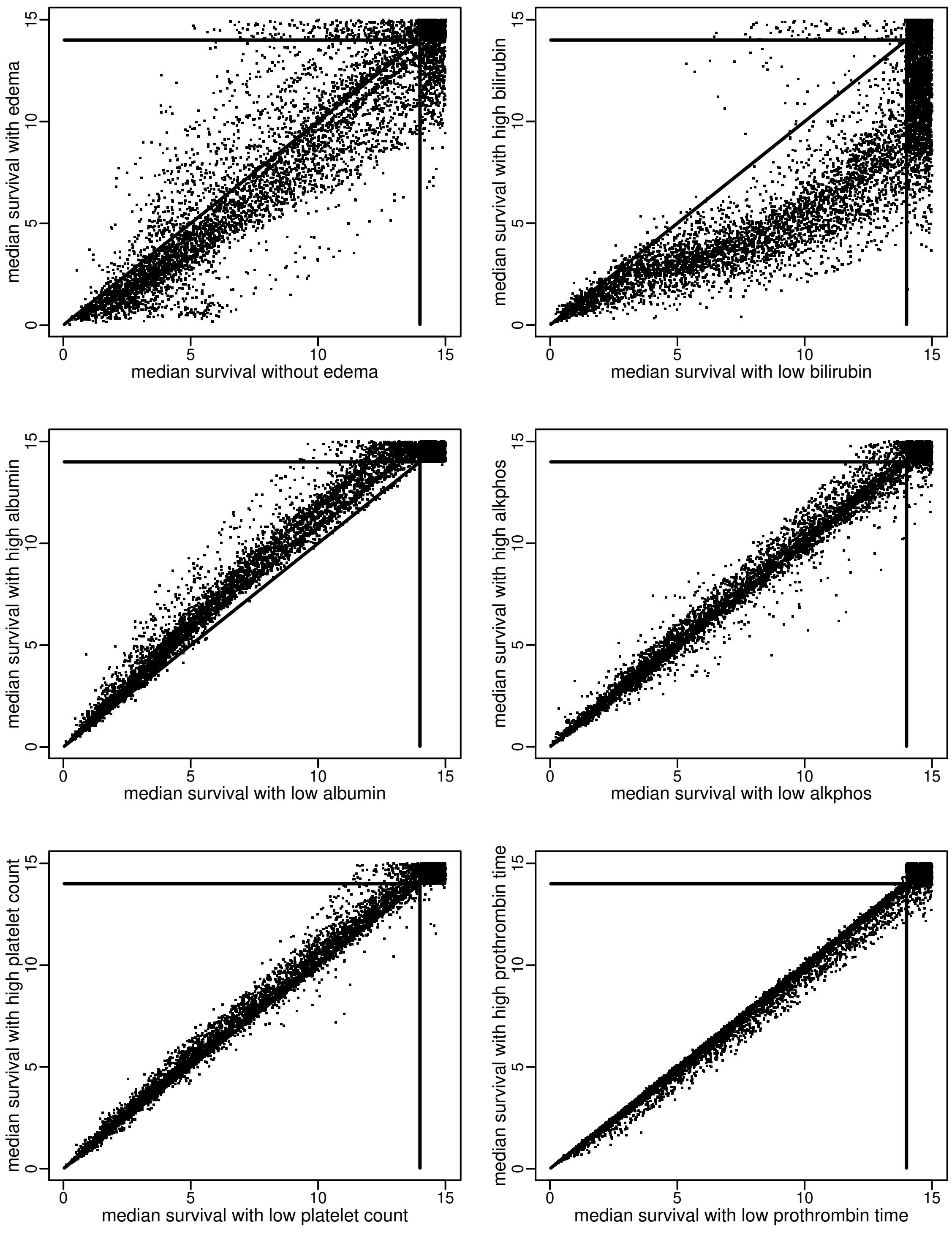
Posterior Distributions of Effects

Using 25 networks from the posterior of model `nnn`.



Posterior Distributions of Effects

Using 25 networks from the posterior of model `nnn`.



Conclusions

- Neural network survival analysis can go beyond simple proportional hazards models. Overfitting can be avoided by using Bayesian methods.

- The posterior distribution can be interpreted even for complex models.

- However: Inference for these models is computationally demanding. The MCMC runs take several hours to a day.

- How does the approach I take of explicitly modelling the baseline hazard compare with using a partial likelihood, or with other neural network approaches?

  (Eg, Faraggi and Simon 1995; Ripley and Ripley 1998, Bakker, Kappen, and Heskes 2000)

Software

Neural network survival analysis models are part of my "software for flexible Bayesian modeling" (for Unix/Linux, not Windows).

This software is freely-available for research purposes from my web site:

`http://www.cs.utoronto.ca/~radford`

The results shown in this talk were obtained using some new features not yet publicly released. A new version with these features will be put on the web site soon.