Neural Networks

Lecture 10
The Bayesian way to fit models

The Bayesian framework

- The Bayesian framework assumes that we always have a prior distribution for everything.
  - The prior may be very vague.
  - When we see some data, we combine our prior distribution with a likelihood term to get a posterior distribution.
  - The likelihood term takes into account how probable the observed data is given the parameters of the model.
    - It favors parameter settings that make the data likely.
    - It fights the prior
    - With enough data the likelihood terms always win.

A coin tossing example

- Suppose we know nothing about coins except that each tossing event produces a head with some unknown probability p and a tail with probability 1-p. Our model of a coin has one parameter, p.
- Suppose we observe 100 tosses and there are 53 heads. What is p?
- The frequentist answer: Pick the value of p that makes the observation of 53 heads and 47 tails most probable.

$$P(D) = p^{53}(1-p)^{47} \leftarrow$$ probability of a particular sequence

$$\frac{dP(D)}{dp} = 53p^{52}(1-p)^{47} - 47p^{53}(1-p)^{46}$$

$$= \left(\frac{53}{p} - \frac{47}{1-p}\right)\left[p^{53}(1-p)^{47}\right]$$

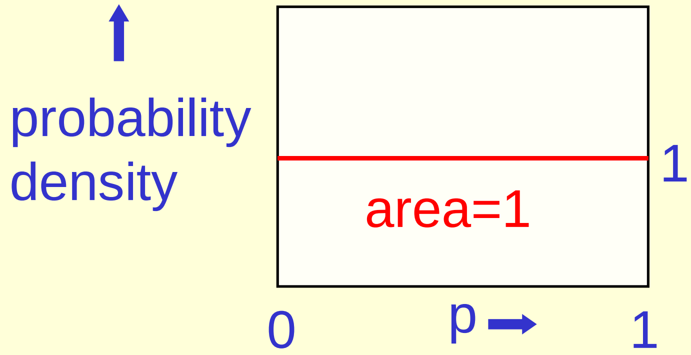
$$= 0 \quad if \quad p = .53$$

Some problems with picking the parameters that are most likely to generate the data

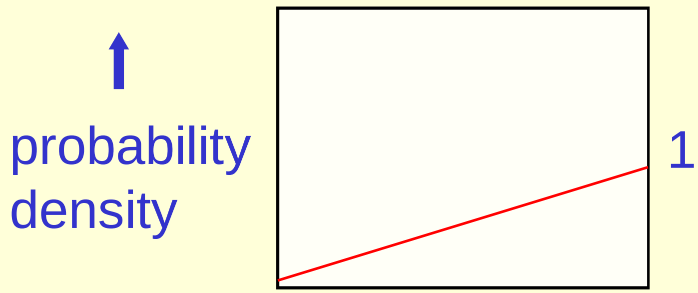
- What if we only tossed the coin once and we got 1 head?
  - Is p=1 a sensible answer?
    - Surely p=0.5 is a much better answer.
- Is it reasonable to give a single answer?
  - If we don't have much data, we are unsure about p.
  - Our computations of probabilities will work much better if we take this uncertainty into account.

Using a distribution over parameter values

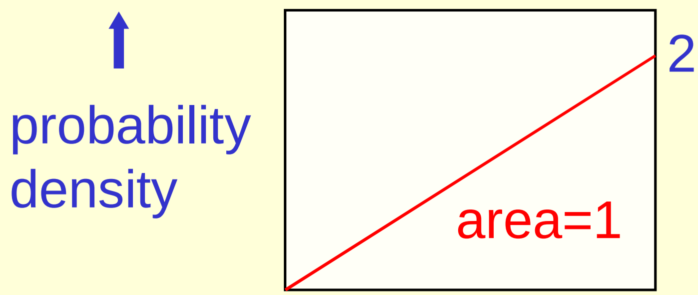
- Start with a prior distribution over p. In this case we used a uniform distribution.



- Multiply the prior probability of each parameter value by the probability of observing a head given that value.

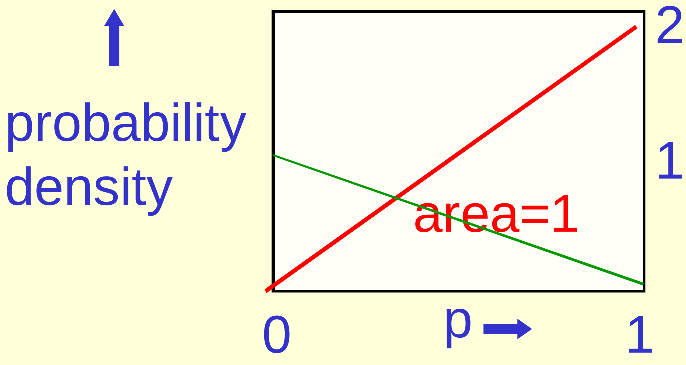


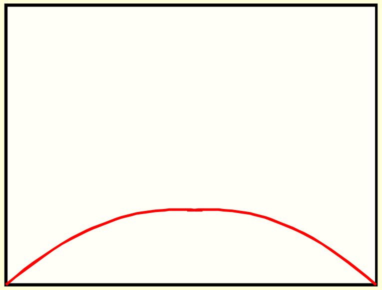
- Then scale up all of the probability densities so that their integral comes to 1. This gives the posterior distribution.

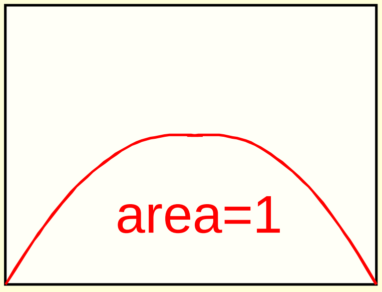


Lets do it again: Suppose we get a tail

- Start with a prior distribution over p.
- Multiply the prior probability of each parameter value by the probability of observing a tail given that value.
- Then renormalize to get the posterior distribution. Look how sensible it is!

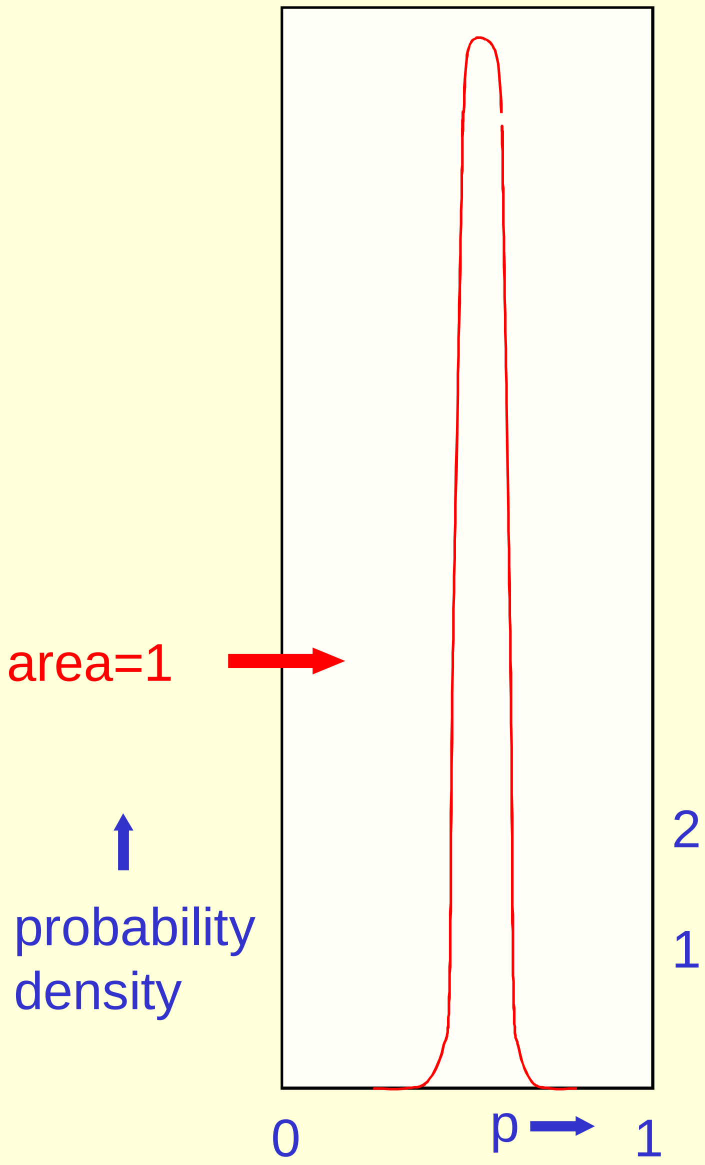






Lets do it another 98 times

- After 53 heads and 47 tails we get a very sensible posterior distribution that has its peak at 0.53 (assuming a uniform prior).



Bayes Theorem

joint probability

conditional probability

$$p(D)\,p(W \mid D) = p(D, W) = p(W)\,p(D \mid W)$$

Prior probability of weight vector W

Probability of observed data given W

Posterior probability of weight vector W given training data D

$$\int_W p(W)\,p(D \mid W)$$

A cheap trick to avoid computing the posterior probabilities of all weight vectors

- Suppose we just try to find the most probable weight vector.
  - We can do this by starting with a random weight vector and then adjusting it in the direction that improves p(W | D).
- It is easier to work in the log domain. If we want to minimize a cost we use negative log probabilities:

$$p(W \mid D) \;=\; p(W) \quad p(D \mid W) \;\; / \;\; p(D)$$

$$Cost = -\log\, p(W \mid D) \;=\; -\log p(W) - \log p(D \mid W) + \log p(D)$$

Why we maximize sums of log probs

- We want to maximize the product of the probabilities of the outputs on all the different training cases
  - Assume the output errors on different training cases, c, are independent.

$$p(D \mid W) = \prod_{c} p(d_c \mid W)$$

- Because the log function is monotonic, it does not change where the maxima are. So we can maximize sums of log probabilities

$$\log p(D \mid W) = \sum_{c} \log p(d_c \mid W)$$

A even cheaper trick

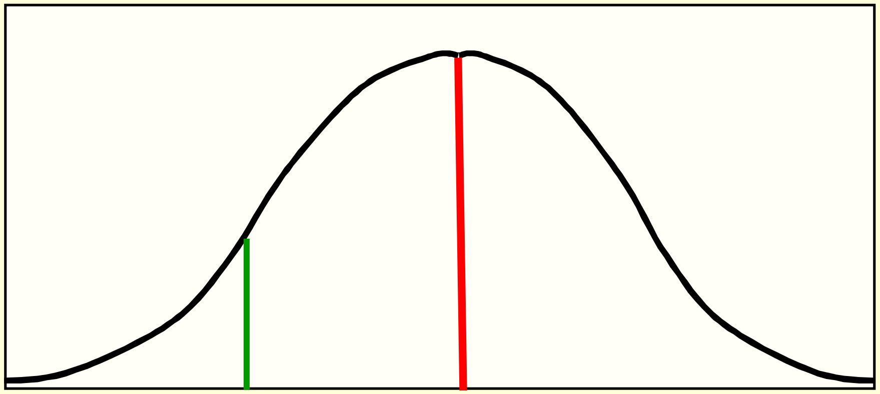
- Suppose we completely ignore the prior over weight vectors
  - This is equivalent to giving all possible weight vectors the same prior probability density.
- Then all we have to do is to maximize:

$$\log p(D \mid W) = \sum_c \log p(D_c \mid W)$$

- This is called maximum likelihood learning. It is very widely used for fitting models in statistics.

Supervised Maximum Likelihood Learning

- Minimizing the squared residuals is equivalent to maximizing the log probability of the correct answer under a Gaussian centered at the model's guess.



d = the correct answer

y = model's estimate of most probable value

$$y_c = f(input_c , W)$$

$$p(output = d_c \mid input_c, W) = p(d_c \mid y_c) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(d_c - y_c)^2}{2\sigma^2}}$$

$$-\log p(output = d_c \mid input_c, W) = k + \frac{(d_c - y_c)^2}{2\sigma^2}$$

Supervised Maximum Likelihood Learning

- Finding a set of weights, W, that minimizes the squared errors is exactly the same as finding a W that maximizes the log probability that the model would produce the desired outputs on all the training cases.
  - We implicitly assume that zero-mean Gaussian noise is added to the model's actual output.
  - We do not need to know the variance of the noise because we are assuming it's the same in all cases. So it just scales the squared error.