COMP9444
Neural Networks and Deep Learning

2a. Probability and Backprop Variations

Textbook, Sections 3.1-3.5, 3.9-3.13, 5.2.2, 5.5, 8.3

Outline

- Probability and Random Variables (3.1-3.2)
- Probability for Continuous Variables (3.3)
- Gaussian Distribution (3.9.3)
- Conditional Probability (3.5)
- Bayes' Rule (3.11)
- Entropy and KL-Divergence (3.13)
- Cross Entropy (3.13)
- Maximum Likelihood (5.5)
- Weight Decay (5.2.2)
- Momentum (8.3)

Probability (3.1)

Begin with a set Ω – the sample space (e.g. 6 possible rolls of a die)

Each $\omega \in \Omega$ is a sample point/possible world/atomic event

A probability space or probability model is a sample space
with an assignment $P(\omega)$ for every $\omega \in \Omega$ such that

$$0 \leq P(\omega) \leq 1$$

$$\sum_{\omega} P(\omega) = 1$$

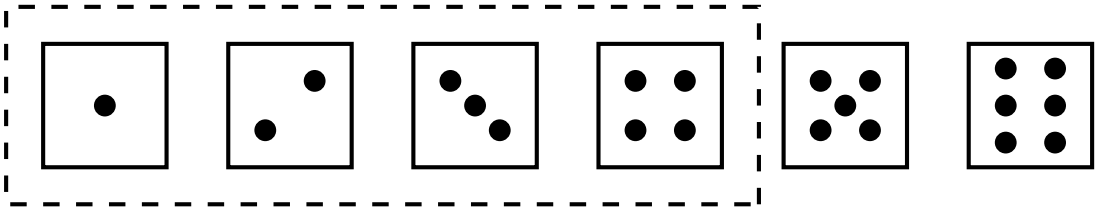
e.g. $P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = \frac{1}{6}$.

Random Events

A random event A is any subset of Ω

$$P(A) = \sum_{\{\omega \in A\}} P(\omega)$$

e.g. $P(\text{die roll} < 5) = P(1) + P(2) + P(3) + P(4) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{2}{3}$



Random Variables (3.2)

A random variable is a function from sample points to some range (e.g. the Reals or Booleans)

For example, `Odd`(3) = `true`

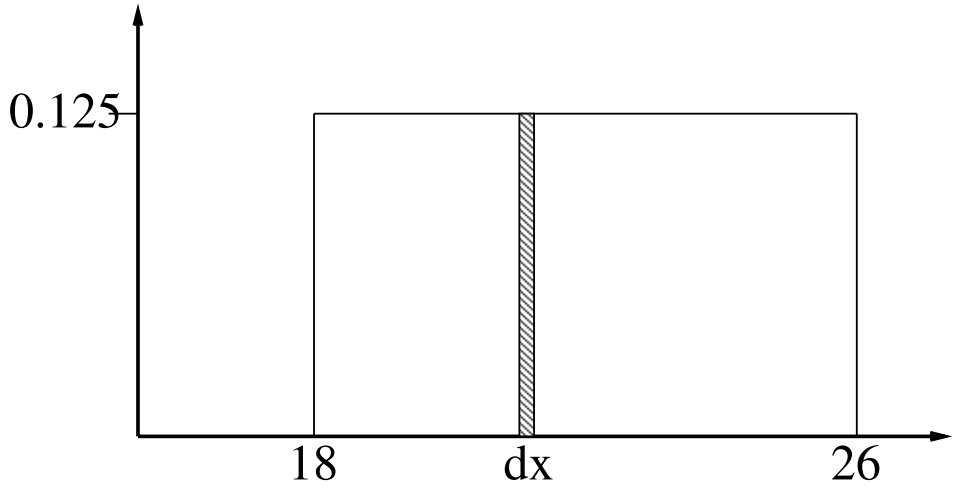
P induces a probability distribution for any random variable X:

$$P(X = x_i) = \sum_{\{\omega : X(\omega) = x_i\}} P(\omega)$$

e.g., $P(\texttt{Odd} = \texttt{true}) = P(1) + P(3) + P(5) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}$

Probability for Continuous Variables (3.3)

e.g. $P(X = x) = U[18, 26](x)$ = uniform density between 18 and 26



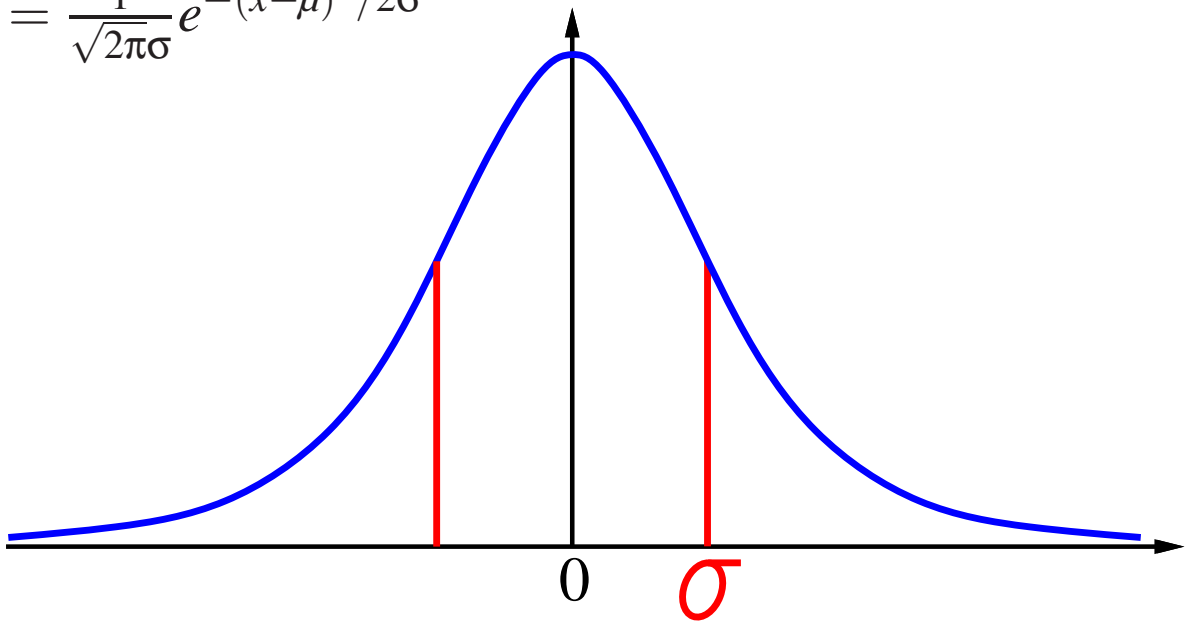
Here P is a density; it integrates to 1.

$P(X = 20.5) = 0.125$ really means

$$\lim_{dx \to 0} P(20.5 \leq X \leq 20.5 + dx)/dx = 0.125$$

Gaussian Distribution (3.9.3)

$$P_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$

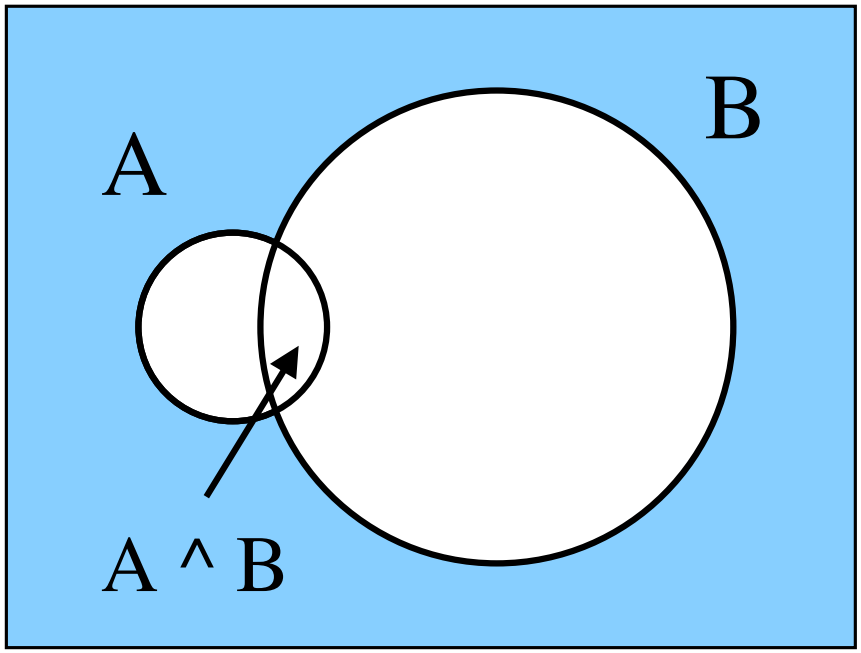


μ = mean

σ = standard deviation

Multivariate Gaussian: $P_{\mu,\sigma}(x) = \prod_i P_{\mu_i,\sigma_i}(x_i)$

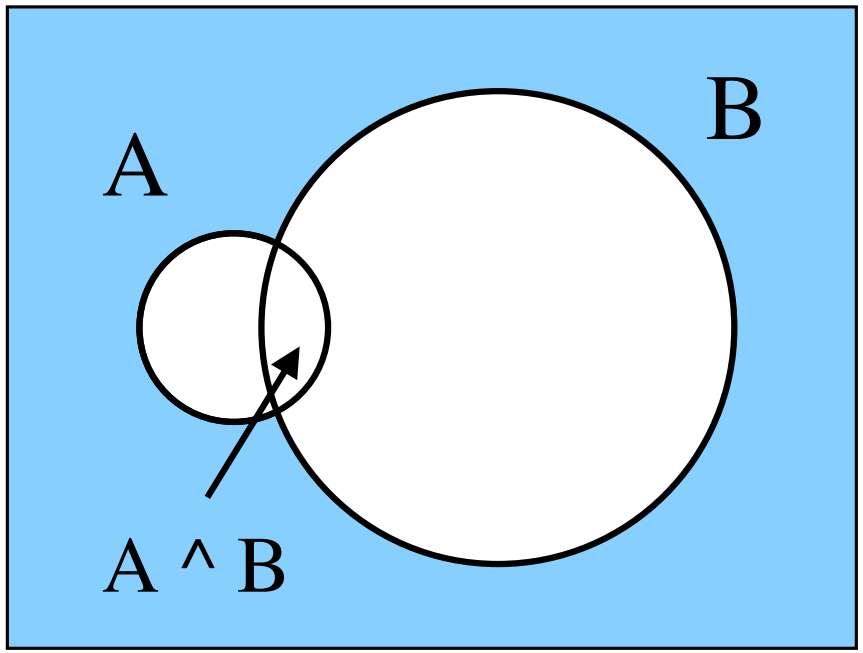
Probability and Logic



Logically related events must have related probabilities

For example, $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

Conditional Probability (3.5)



If $P(B) \neq 0$, then the conditional probability of A given B is

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$

Bayes' Rule (3.11)

The formula for conditional probability can be manipulated to find a relationship when the two variables are swapped:

$$P(A \wedge B) = P(A \mid B)P(B) = P(B \mid A)P(A)$$

$$\rightarrow \text{Bayes' rule} \quad P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

This is often useful for assessing the probability of an underlying cause after an effect has been observed:

$$P(\text{Cause}|\text{Effect}) = \frac{P(\text{Effect}|\text{Cause})P(\text{Cause})}{P(\text{Effect})}$$

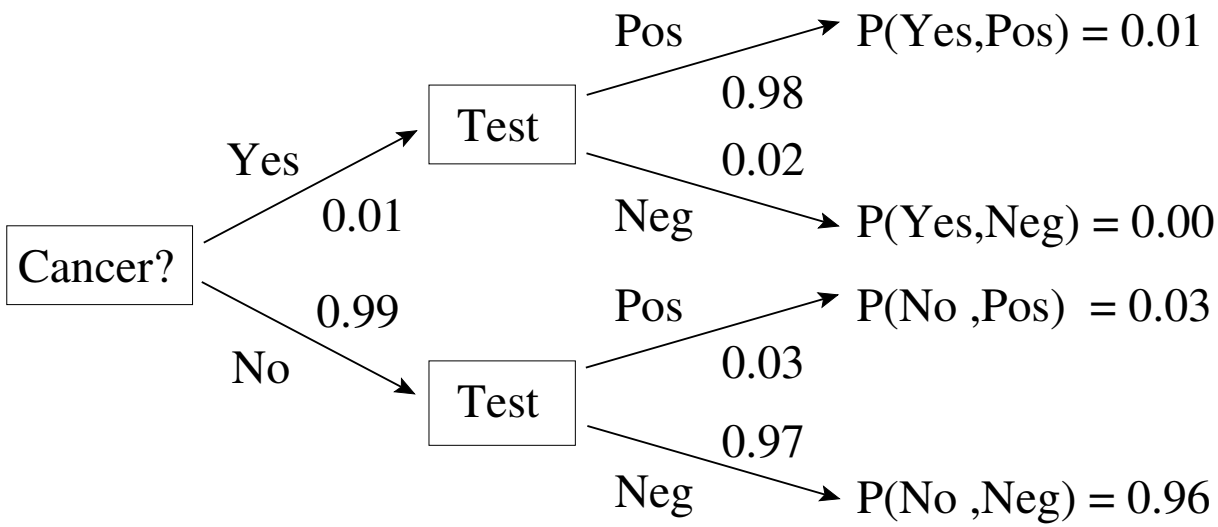
Example: Medical Diagnosis

Question: Suppose we have a test for a type of cancer which occurs in 1% of patients. The test has a sensitivity of 98% and a specificity of 97%.
If a patient tests positive, what is the probability that they have the cancer?

Answer: There are two random variables: Cancer (true or false) and Test (positive or negative). The probability is called a prior, because it represents our estimate of the probability before we have done the test (or made some other observation). The sensitivity and specificity are interpreted as follows:

$$P(\texttt{positive} \,|\, \texttt{cancer}) = 0.98, \quad \text{and} \quad P(\texttt{negative} \,|\, \neg\texttt{cancer}) = 0.97$$

Bayes' Rule for Medical Diagnosis



$$P(\texttt{cancer}\,|\,\texttt{positive}) = \frac{P(\texttt{positive}\,|\,\texttt{cancer})P(\texttt{cancer})}{P(\texttt{positive})}$$

$$= \frac{0.98 * 0.01}{0.98 * 0.01 + 0.03 * 0.99} = \frac{0.01}{0.01 + 0.03} = \frac{1}{4}$$

Example: Light Bulb Defects

Question: You work for a lighting company which manufactures 60% of its light bulbs in Factory A and 40% in Factory B. One percent of the light bulbs from Factory A are defective, while two percent of those from Factory B are defective. If a random light bulb turns out to be defective, what is the probability that it was manufactured in Factory A?

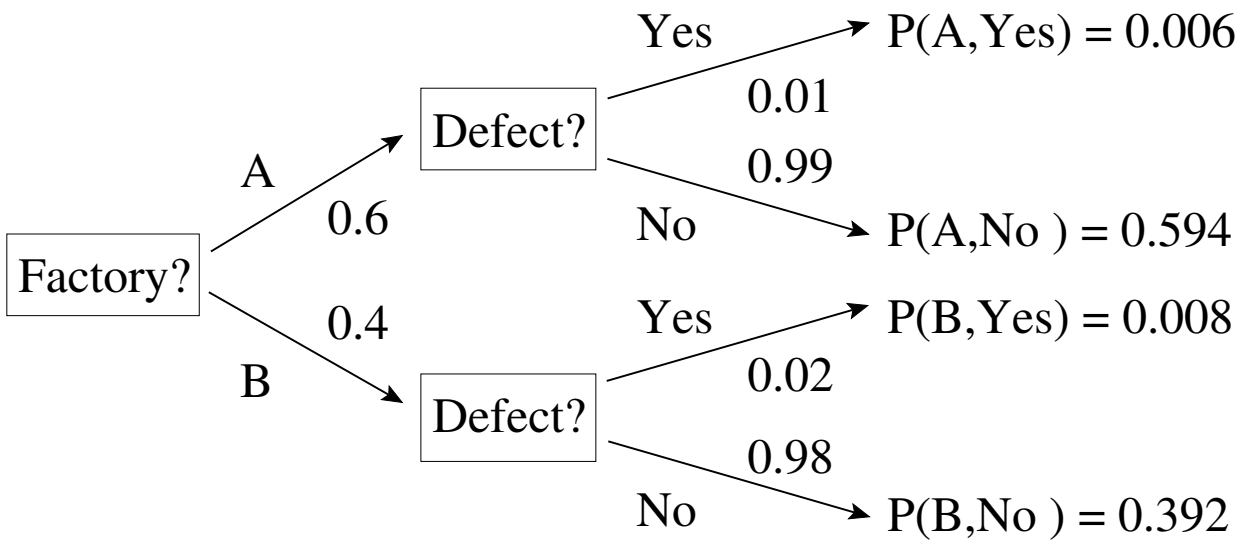
Answer: There are two random variables: Factory (A or B) and Defect (Yes or No). In this case, the prior is:

$P(A) = 0.6, \qquad P(B) = 0.4$

The conditional probabilities are:

$P(\texttt{defect} \,|\, \texttt{A}) = 0.01, \quad \text{and} \quad P(\texttt{defect} \,|\, \texttt{B}) = 0.02$

Bayes' Rule for Light Bulb Defects



$$P(\mathtt{A}\,|\,\mathtt{defect}) = \frac{P(\mathtt{defect}\,|\,\mathtt{A})P(\mathtt{A})}{P(\mathtt{defect})}$$

$$= \frac{0.01 * 0.6}{0.01 * 0.6 + 0.02 * 0.4} = \frac{0.006}{0.006 + 0.008} = \frac{3}{7}$$

Entropy and KL-Divergence (3.13)

The entropy of a discrete probability distribution $p = \langle p_1, \ldots, p_n \rangle$ is

$$H(p) = \sum_{i=1}^{n} p_i \, (-\log_2 p_i)$$

Given two probability distributions $p = \langle p_1, \ldots, p_n \rangle$ and $q = \langle q_1, \ldots, q_n \rangle$ on the same set Ω, the Kullback-Leibler Divergence between p and q is

$$D_{\mathrm{KL}}(p \,\|\, q) = \sum_{i=1}^{n} p_i \, (\log_2 p_i - \log_2 q_i)$$

KL-Divergence is like a "distance" from one probability distribution to another. But, it is not symmetric.

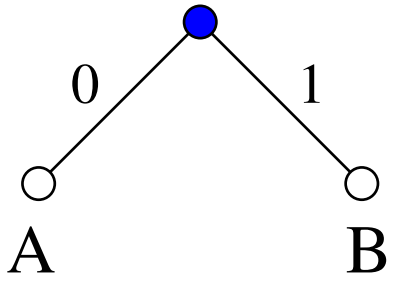
$$D_{\mathrm{KL}}(p \,\|\, q) \;\neq\; D_{\mathrm{KL}}(q \,\|\, p)$$

Entropy and Huffmann Coding

Entropy is the number of bits per symbol achieved by a (block) Huffman Coding scheme.

Example 1: $H(\langle 0.5, 0.5 \rangle) = 1$ bit.

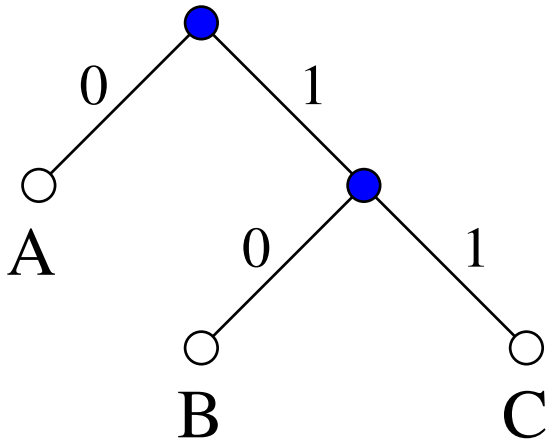
Suppose we want to encode, in zeros and ones, a long message composed of the two letters `A` and `B`, which occur with equal frequency. This can be done efficiently by assigning `A=0`, `B=1`. In other words, one bit is needed to encode each letter.



Entropy and Huffmann Coding

Example 2: $H(\langle 0.5, 0.25, 0.25 \rangle) = 1.5$ bits.

Suppose we need to encode a message consisting of the letters `A`, `B` and `C`, and that `B` and `C` occur equally often but `A` occurs twice as often as the other two letters. In this case, the most efficient code would be `A=0`, `B=10`, `C=11`. The average number of bits needed to encode each letter is 1.5.



Entropy and KL-Divergence

If the samples occur in some other proportion, we would need to "block" them together in order to encode them efficiently. But, the average number of bits required by the most efficient coding scheme is given by

$$H(\langle p_1, \ldots, p_n \rangle) = \sum_{i=1}^{n} p_i \, (-\log_2 p_i)$$

$D_{KL}(q \,\|\, p)$ is the number of extra bits we need to trasmit if we designed a code for $q()$ but it turned out that the samples were drawn from $p()$ instead.

$$D_{KL}(p \,\|\, q) = \sum_{i=1}^{n} p_i \, (\log_2 p_i - \log_2 q_i)$$

Continuous Entropy and KL-Divergence

- the entropy of a continuous distribution $p()$ is

$$H(p) = \int_{\theta} p(\theta)(-\log p(\theta))\, d\theta$$

- for a multivariate Gaussian distribution,

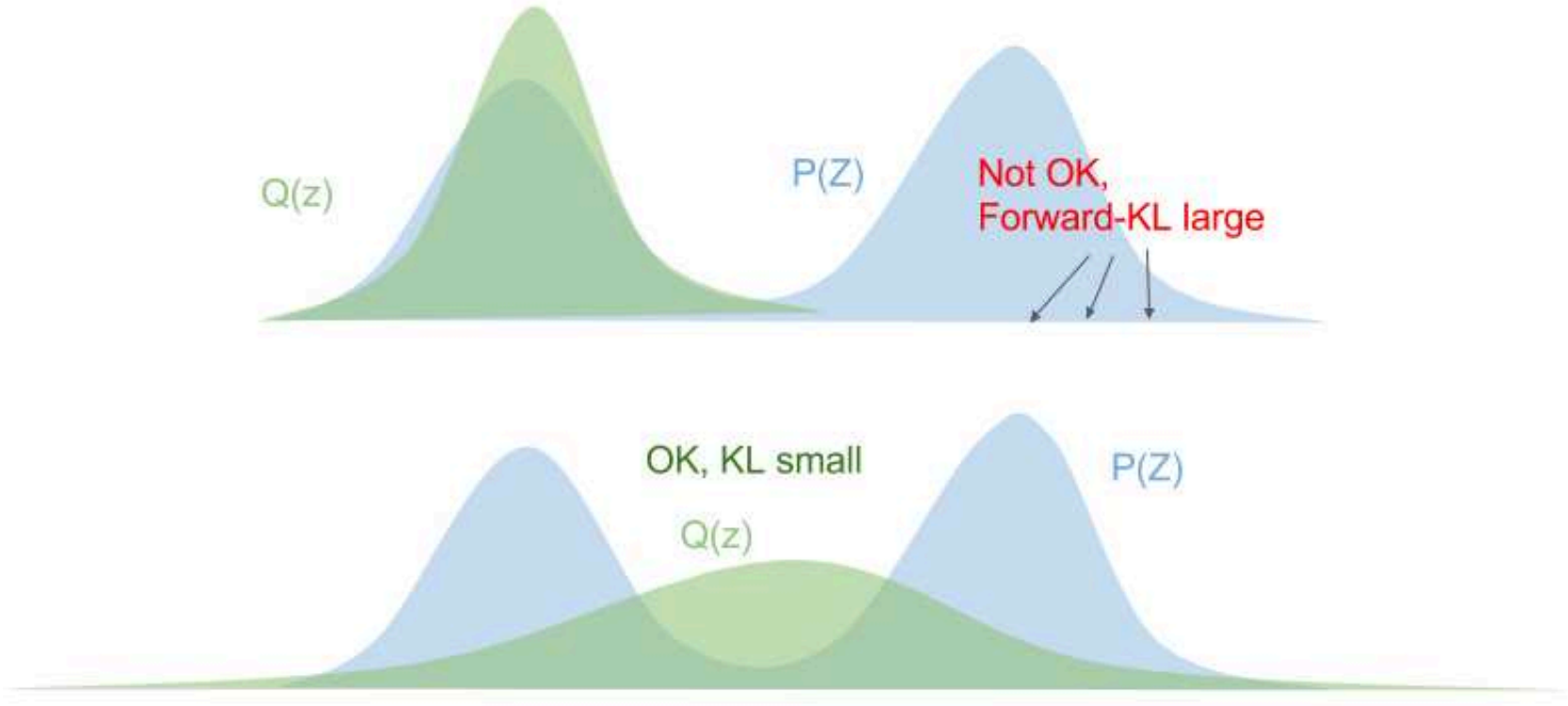
$$H(p) = \sum_{i} \log \sigma_i$$

- KL-Divergence

$$D_{KL}(p \,\|\, q) = \int_{\theta} p(\theta)(\log p(\theta) - \log q(\theta))\, d\theta$$

Forward KL-Divergence

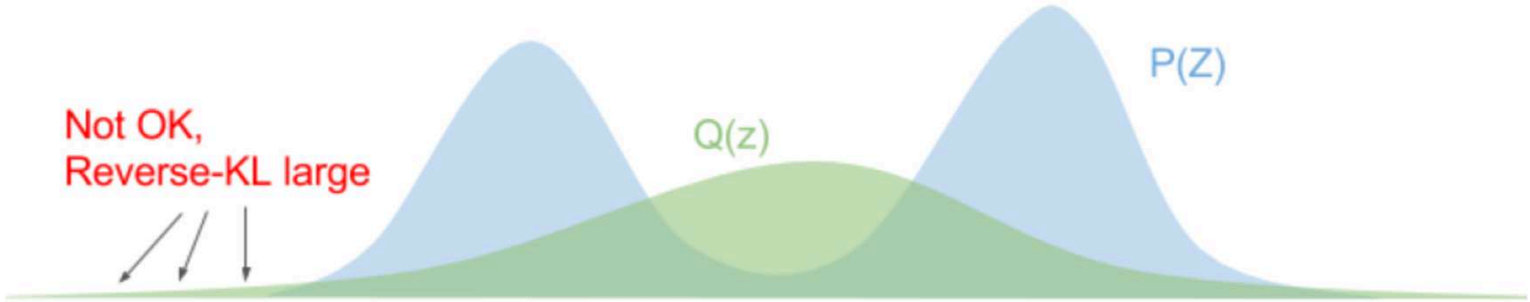
Given P, choose Gaussian Q to minimize $\mathrm{D}_{\mathrm{KL}}(P \,\|\, Q)$

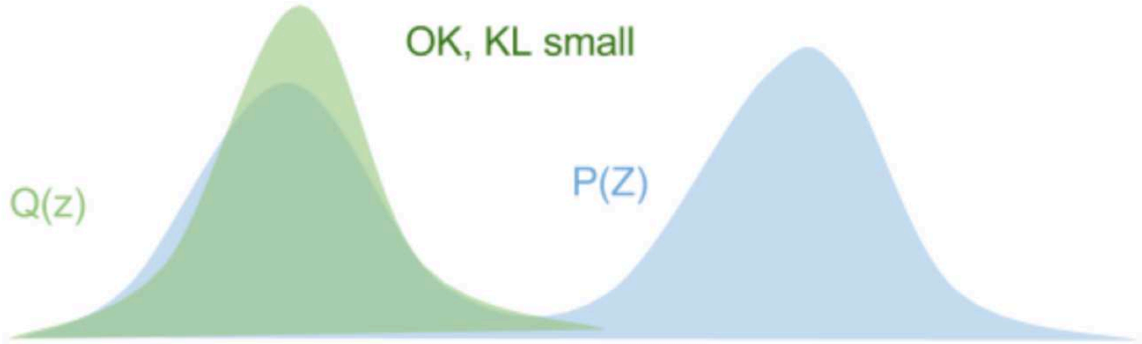


Q must not be small in any place where P is large.

Reverse KL-Divergence

Given P, choose Gaussian Q to minimize $\mathrm{D}_{\mathrm{KL}}(Q \,\|\, P)$





Q just needs to be concentrated in some place where P is large.

KL-Divergence in Deep Learning

- KL-Divergence is used in a number of deep learning algorithms, including Variational Autoencoders (Week 10)
- KL-Divergence is also used in some policy-based deep reinforcement learning algorithms such as Variational Inference or Trust Region Policy Optimization (TPRO) (but we will not cover these in detail)

Variations on Backprop

- Cross Entropy
  - problem: least squares error function not suitable for classification, where target = 0 or 1
  - mathematical theory: maximum likelihood
  - solution: replace with cross entropy error function
- Weight Decay
  - problem: weights "blow up", and inhibit further learning
  - mathematical theory: Bayes' rule
  - solution: add weight decay term to error function
- Momentum
  - problem: weights oscillate in a "rain gutter"
  - solution: weighted average of gradient over time

Cross Entropy

For classification tasks, target t is either 0 or 1, so better to use

$$E = -t\log(z) - (1-t)\log(1-z)$$

This can be justified mathematically, and works well in practice – especially when negative examples vastly outweigh positive ones.
It also makes the backprop computations simpler

$$\frac{\partial E}{\partial z} = \frac{z-t}{z(1-z)}$$

$$\text{if} \quad z = \frac{1}{1+e^{-s}},$$

$$\frac{\partial E}{\partial s} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial s} = z - t$$

Maximum Likelihood (5.5)

H is a class of hypotheses

$P(D\,|\,h)$ = probability of data D being generated under hypothesis $h \in H$.

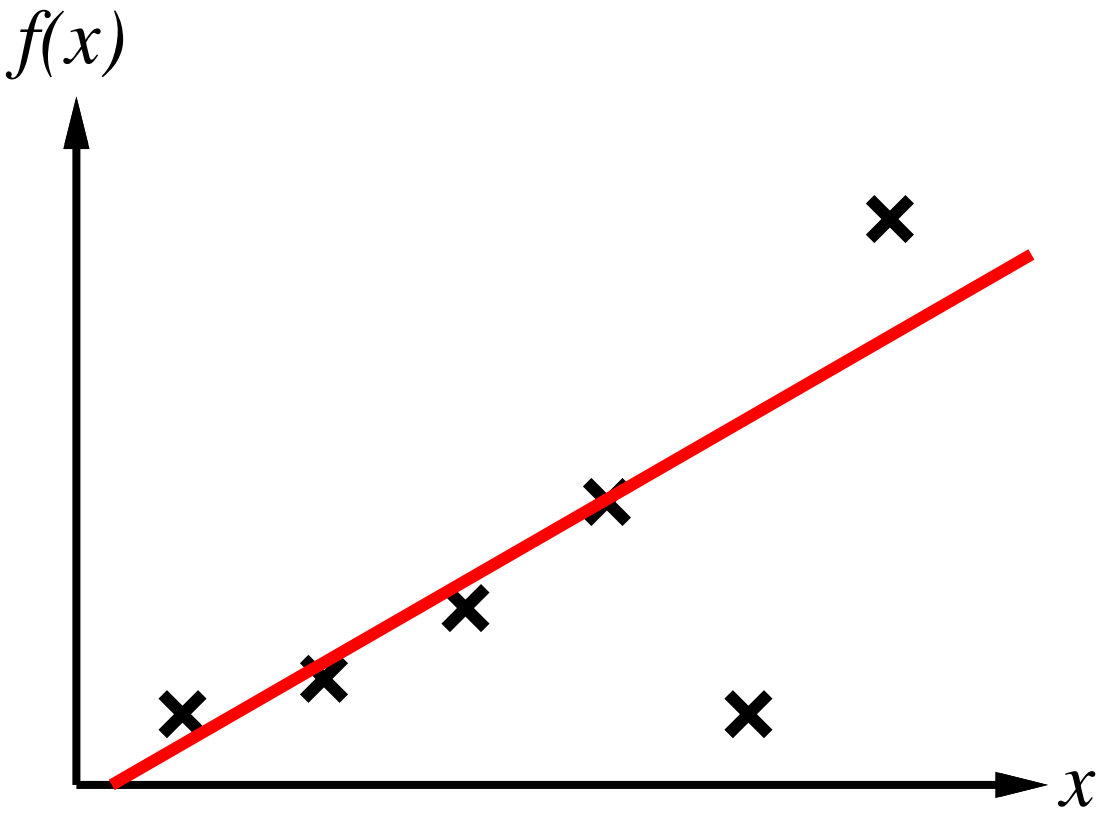
$\log P(D\,|\,h)$ is called the likelihood.

ML Principle: Choose $h \in H$ which maximizes this likelihood,

i.e. maximizes $P(D\,|\,h)$ [or, maximizes $\log P(D\,|\,h)$]

In our case, each hypothesis h is a choice of parameters for a neural network or other function applied to input values x while the observed data D consist of the target values t.

Least Squares Line Fitting



Derivation of Least Squares

Suppose the data are generated by a linear function $f()$ plus Gaussian noise with mean zero and standard deviation σ.

$$
\begin{aligned}
P(D|h) = P(t|f) &= \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(t_i - f(x_i))^2} \\
\log P(t|f) &= \sum_i -\frac{1}{2\sigma^2}(t_i - f(x_i))^2 - \log(\sigma) - \frac{1}{2}\log(2\pi) \\
f_{ML} &= \text{argmax}_{f \in H} \log P(t|f) \\
&= \text{argmin}_{f \in H} \sum_i (t_i - f(x_i))^2
\end{aligned}
$$

(Note: we do not need to know σ)

Derivation of Cross Entropy (3.9.1)

For classification tasks, t is either 0 or 1.

Assume t is generated by hypothesis f as follows:

$$\begin{aligned} P(1\,|\,f(x_i)) &= f(x_i) \\ P(0\,|\,f(x_i)) &= (1-f(x_i)) \\ \text{i.e.} \quad P(t_i\,|\,f(x_i)) &= f(x_i)^{t_i}(1-f(x_i))^{(1-t_i)} \end{aligned}$$

then

$$\begin{aligned} \log P(t|f) &= \sum_i t_i \log f(x_i) + (1-t_i)\log(1-f(x_i)) \\ f_{ML} &= \text{argmax}_{f\in H} \sum_i t_i \log f(x_i) + (1-t_i)\log(1-f(x_i)) \end{aligned}$$

(Can also be generalized to multiple classes.)

Weight Decay (5.2.2)

Sometimes we add a penalty term to the loss function which encourages the neural network weights w_j to remain small:

$$E = \frac{1}{2}\sum_i (z_i - t_i)^2 + \frac{\lambda}{2}\sum_j w_j^2$$

This can prevent the weights from "saturating" to very high values.

It is sometimes referred to as "elastic weights" because the weights experience a force as if there were a spring pulling them back towards the origin according to Hooke's Law.

The scaling factor λ needs to be determined from experience, or empirically.

Bayesian Inference

H is a class of hypotheses

$P(D|h) =$ probability of data D being generated under hypothesis $h \in H$.

$P(h|D) =$ probability that h is correct, given that data D were observed.

Bayes' Theorem:

$$P(h|D)P(D) = P(D|h)P(h)$$

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

$P(h)$ is called the prior.

Weight Decay as MAP Estimation (5.6.1)

We assume a Gaussian prior distribution for the weights, i.e.

$$P(w) = \prod_j \frac{1}{\sqrt{2\pi}\sigma_0} e^{-w_j^2/2\sigma_0^2}$$

Then

$$P(w|t) = \frac{P(t|w)P(w)}{P(t)} = \frac{1}{P(t)} \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(z_i - t_i)^2} \prod_j \frac{1}{\sqrt{2\pi}\sigma_0} e^{-w_j^2/2\sigma_0^2}$$

$$\log P(w|t) = -\frac{1}{2\sigma^2} \sum_i (z_i - t_i)^2 - \frac{1}{2\sigma_0^2} \sum_j w_j^2 + \text{constant}$$

$$w_{\text{MAP}} = \text{argmax}_{w \in H} \log P(w|t)$$

$$= \text{argmin}_{w \in H} \left(\frac{1}{2} \sum_i (z_i - t_i)^2 + \frac{\lambda}{2} \sum_j w_j^2 \right)$$

where $\lambda = \sigma_0^2/\sigma^2$.

Momentum (8.3)

雨水槽 震荡

If landscape is shaped like a "rain gutter", weights will tend to oscillate without much improvement.

Solution: add a momentum factor

$$\begin{aligned} \delta w &\leftarrow \alpha\, \delta w - \eta \frac{\partial E}{\partial w} \\ w &\leftarrow w + \delta w \end{aligned}$$

Hopefully, this will dampen sideways oscillations but amplify downhill motion by $\frac{1}{1-\alpha}$.

普通梯度下降： w = w - learning_rate * dw
借助动量参数M： v = M*v - learning_rate * dw
w = w + v

其中，v初始化为0，M是设定的一个超变量，最常见的设定值是0.9。可以这样理解上式：如果上次的momentum()与这次的负梯度方向是相同的，那这次下降的幅度就会加大，从而加速收敛。