COMP9444
Neural Networks and Deep Learning
Term 2, 2025

Week 3 Tutorial Solutions

This page was last updated: 06/20/2025 17:07:00

1. **Bayes' Rule**

   One bag contains 2 red balls and 3 white balls. Another bag contains 3 red balls and 2 green balls. One of these bags is chosen at random, and two balls are drawn randomly from that bag, without replacement. Both of the balls turn out to be red. What is the probability that the first bag is the one that was chosen?

   Let B = first bag is chosen, R = both balls are red. Then

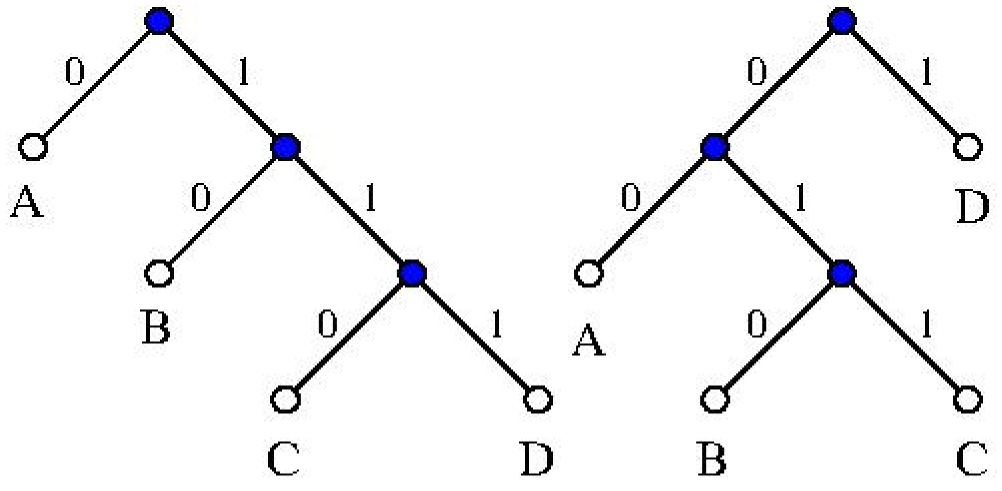
   P (R | B) = (2/5)*(1/4) = 1/10
   P (R | ¬B) = (3/5)*(2/4) = 3/10
   P (R) = (1/2)*(1/10) + (1/2)*(3/10) = 1/5
   P (B | R) = P(R|B)*P(B) / P(R) = (1/10)*(1/2)/(1/5) = 1/4

2. **Entropy and KL-Divergence for Discrete Distributions**

   Consider two probability distributions on the same space Ω = {A, B, C, D}:

   $p = \langle \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8} \rangle$
   $q = \langle \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{2} \rangle$

   a. Construct a Huffmann tree for each distribution *p* and *q*



Note: the answer is not unique; this is one possible tree in each case.

b. Compute the entropy $H(p)$

$$H(p) = H(q) = \tfrac{1}{2}(-\log \tfrac{1}{2}) + \tfrac{1}{4}(-\log \tfrac{1}{4}) + \tfrac{1}{8}(-\log \tfrac{1}{8}) + \tfrac{1}{8}(-\log \tfrac{1}{8})$$
$$= \tfrac{1}{2}(1) + \tfrac{1}{4}(2) + \tfrac{1}{8}(3) + \tfrac{1}{8}(3) = 1.75$$

c. Compute the KL-Divergence in each direction $D_{KL}(p \| q)$ and $D_{KL}(q \| p)$.

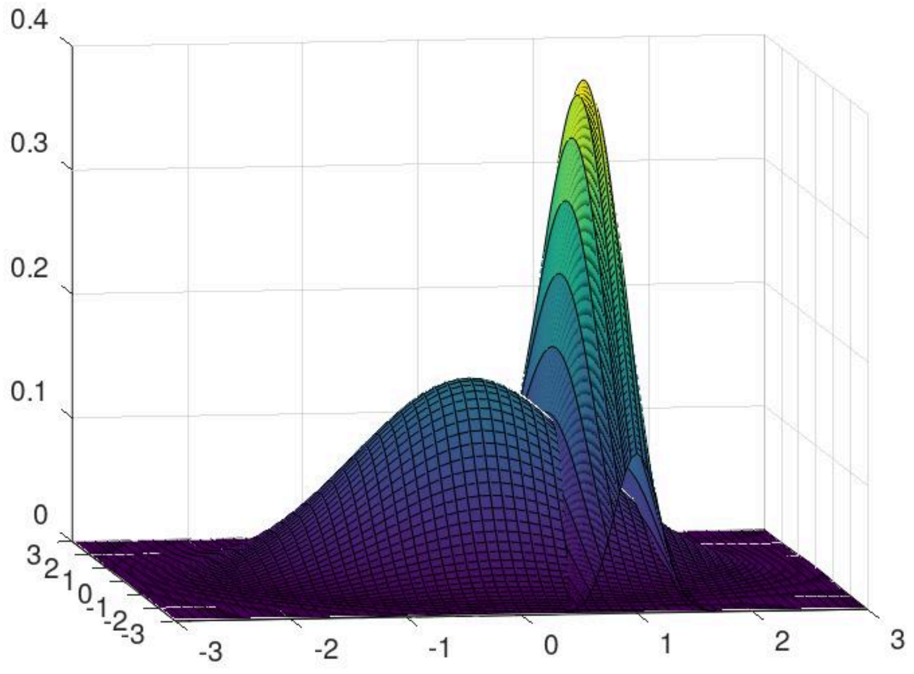
$$D_{KL}(p \| q) = \tfrac{1}{2}(2-1) + \tfrac{1}{4}(3-2) + \tfrac{1}{8}(3-3) + \tfrac{1}{8}(1-3) = 0.5$$

$$D_{KL}(q \| p) = \tfrac{1}{4}(1-2) + \tfrac{1}{8}(2-3) + \tfrac{1}{8}(3-3) + \tfrac{1}{2}(3-1) = 0.625$$

Which one is larger? Why?

$D_{KL}(q \| p)$ is larger, mainly because the frequency of D has increased from ⅛ to ½, so it incurs a cost of 3–1=2 additional bits every time it occurs (which is often).

3. **Entropy, KL-Divergence and W_2 Distance for Bivariate Gaussians**



Consider two bivariate Gaussian distributions p and q.

q has mean $\mu_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and variance $\Sigma_1 = \begin{bmatrix} 0.04 & 0 \\ 0 & 4 \end{bmatrix}$

p has mean $\mu_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and variance $\Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

a. Compute the Entropy $\mathrm{H}(p)$ and $\mathrm{H}(q)$. Which one is larger? Why?

$\mathrm{H}(p) = ½\log|\Sigma_2| + 1 + \log(2\pi) = 1 + \log(2\pi) \approx 2.8379$

$\mathrm{H}(q) = ½\log|\Sigma_1| + 1 + \log(2\pi) = \log(0.4) + 1 + \log(2\pi) \approx 1.9216$

When compared to *p, q* has been compressed by a factor of 5 in one direction but stretched by a factor of 2 in the other, resulting in an overall compression factor of 0.4. Consequently, H(*q*) is smaller than H(*p*) because the probability distribution is more concentrated and therefore less "uncertain".
H(*p*) is larger than H(*q*) by $-\log(0.4) = \log(2.5) \approx 0.9163$

b. Compute the KL-Divergence in each direction $\mathrm{D}_{\mathrm{KL}}(q \,||\, p)$ and $\mathrm{D}_{\mathrm{KL}}(p \,||\, q)$. Which one is larger? Why?

$\mathrm{D}_{\mathrm{KL}}(q \,||\, p) = ½[||\mu_1||^2 + \mathrm{Trace}(\Sigma_1) - \log|\Sigma_1| - d]$
$= ½[1 + 4.04 - \log(0.16) - 2] = 1.52 - \log(0.4) \approx 2.4363$

$\mathrm{D}_{\mathrm{KL}}(p \,||\, q) = ½[(\mu_1-\mu_2)^T\Sigma_1^{-1}(\mu_1-\mu_2) + \mathrm{Trace}(\Sigma_1^{-1}\Sigma_2) + \log|\Sigma_1| - \log|\Sigma_2| - d]$
$= ½[25 + 25.25 + \log(0.16) - \log(1) - 2] = 24.125 + \log(0.4) \approx 23.2087$

The second one is much larger, because there are places where *p* is large but *q* is very close to zero.

c. Compute the Wasserstein Distance $\mathrm{W}_2(q, p)$

$\mathrm{W}_2(q, p)^2 = ||\mu_1||^2 + \Sigma_i(\sigma_i - 1)^2$
$= 1 + (0.2 - 1)^2 + (2 - 1)^2 = 1 + 0.64 + 1 = 2.64$

Alternatively,
$\mathrm{W}_2(q, p)^2 = ||\mu_1 - \mu_2||^2 + \mathrm{Trace}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1\Sigma_2)^{½})$
$= 1 + 4.04 + 2 - 2(2.2) = 2.64$

$\mathrm{W}_2(q, p) = \sqrt{2.64} \approx 1.6248$

4. **Simple PyTorch Program to learn XOR**

Copy the following code into a file called `xor.py` and run it.

```
import torch
import torch.utils.data
import torch.nn.functional as F
```

```
lr = 0.1
mom = 0.0
init = 1.0

class MyModel(torch.nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        # define structure of the network here
        self.in_hid  = torch.nn.Linear(2,2)
        self.hid_out = torch.nn.Linear(2,1)
    def forward(self, input):
        # apply network and return output
        hid_sum = self.in_hid(input)
        hidden  = torch.tanh(hid_sum)
        out_sum = self.hid_out(hidden)
        output  = torch.sigmoid(out_sum)
        return(output)

device = 'cpu'

input  = torch.Tensor([[0,0],[0,1],[1,0],[1,1]])
target = torch.Tensor([[0],[1],[1],[0]])

xor_dataset  = torch.utils.data.TensorDataset(input,target)
train_loader = torch.utils.data.DataLoader(xor_dataset,batch_size=4)

# create neural network according to model specification
net = MyModel().to(device) # CPU or GPU

# initialize weight values
net.in_hid.weight.data.normal_(0,init)
net.hid_out.weight.data.normal_(0,init)

# choose between SGD, Adam or other optimizer
optimizer = torch.optim.SGD(net.parameters(),lr=lr,momentum=mom)

epochs = 10000

epoch = 0
loss = 1
while epoch < epochs and loss >= 0.01:
    epoch = epoch + 1
    for batch_id, (data,target) in enumerate(train_loader):
        optimizer.zero_grad() # zero the gradients
        output = net(data)    # apply network
        loss = F.binary_cross_entropy(output,target)
        loss.backward()      # compute gradients
        optimizer.step()     # update weights
        if epoch % 100 == 0:
            print('ep%3d: loss = %7.4f' % (epoch, loss.item()))

if loss < 0.02:
    print("Global Mininum")
else:
    print("Local Minimum")
```

a. Run the code ten times. For how many runs does it reach the Global Minimum? For how many runs does it reach a Local Minimum?

It should reach the Global Minimum in approximately half of runs, and get stuck in a Local Minimum for the other half.

b. Keeping the learning rate fixed at 0.1, adjust the values of momentum (`mom`) on line 6 and initial weight size (`init`) on line 7 to see if you can find values for which the code converges relatively quickly to the Global Minimum on almost every run.

With `mom = 0.9` and `init = 0.01` it should successfully reach the Global Minimum in 99% of runs.

5. **Paper Discussion**

Following Week 2's paper discussion, in Week 3, we will be reading the following paper:

- Sutskever et al., Sequence to Sequence Learning with Neural Networks, NeurIPS, 2014.

This is Seq2Seq (or Encoder-Decoder architecture) paper, one of the seminal papers in the field of Natural Language Processing (NLP) and has revolutioned many applications such as Machine Translation, Text Summarisation, Question Answering, etc. Infact, many multimodal applications such as Image Captioning and Visual Question Answering also use encoder-decoder architecture. You can download paper by clicking on the paper title or by copy-pasting this URL: `https://research.google.com/pubs/archive/43155.pdf`

6. **Any Further Questions**

Any further questions or discussion about PyTorch, other parts of the course, or broader implications of deep learning.