02456 Deep Learning
Written exam
(solutions)

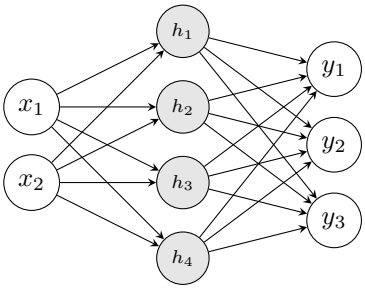
Technical University of Denmark

December 10, 2025

This document contains the 10 exam questions on 7 pages. **All answers must be filled in on the *answer sheet*; do not hand in this document**. Please read the instructions on the *answer sheet* carefully.

Question 1

Consider the fully connected neural network shown below, where $\mathbf{x} \in \mathbb{R}^2$ is the input and $\mathbf{y} \in \mathbb{R}^3$ is the output.



Assume each neuron has a bias term. How many parameters does the model have in total?

A. 7

B. 9

C. 20

D. 27

E. Don't know

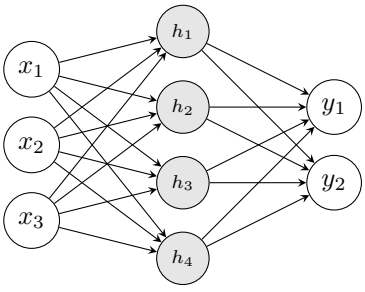
Solution: It has $2 \times 4 + 4 \times 3 = 20$ weights and $4 + 3 = 7$ biases, which is 27 in total.

Question 2

Consider the fully connected neural network with one hidden layer defined as

$$f(\mathbf{x}) = W^{(2)}\sigma(W^{(1)}\mathbf{x}), \tag{1}$$

where $W^{(1)} \in \mathbb{R}^{n \times m}$, $W^{(2)} \in \mathbb{R}^{k \times \ell}$ and σ is the non-linear activation function. Assume that the network has the architecture shown below.



What are the dimensions of the weight matrices?

A. $n = 3$, $m = 4$, $k = 4$, $\ell = 2$

B. $n = 4$, $m = 3$, $k = 2$, $\ell = 4$

C. $n = 2$, $m = 4$, $k = 4$, $\ell = 3$

D. $n = 4$, $m = 2$, $k = 3$, $\ell = 4$

E. Don't know

Solution: From the figure, we have that $\dim(\mathbf{x}) = 3$, $\dim(\mathbf{h}) = 4$ and $\dim(\mathbf{y}) = 2$. The matrix $W^{(1)}$, must therefore have dimensions $\dim(\mathbf{h}) \times \dim(\mathbf{x}) = 4 \times 3$ and accordingly $n = 4$ and $m = 3$. Similarly, $W^{(2)}$ must have dimensions $\dim(\mathbf{y}) \times \dim(\mathbf{h}) = 2 \times 4$ and accordingly $k = 2$ and $\ell = 4$.

Question 3

Consider the following simple neural network with a single hidden layer:

$$h = \text{ReLU}(wx), \qquad y = vh, \qquad L = \frac{1}{2}(y - t)^2, \tag{2}$$

where $x \in \mathbb{R}$ is the input, $h \in \mathbb{R}$ is the hidden layer output, $y \in \mathbb{R}$ is the output, $L \in \mathbb{R}$ is the loss, $t \in \mathbb{R}$ is the target, $w, v \in \mathbb{R}$ are weights, and $\text{ReLU}(z) = \max(0, z)$. Assume that $x = 1$, $t = 6$, $w = 2$, and $v = 4$. What is the value of the gradient of the loss $\frac{\partial L}{\partial w}$ for this input and target?

A. 0

B. 4

C. 8

D. 16

E. Don't know

Solution: First, we evaluate the forward pass:

$$h = \text{ReLU}(wx) = \text{ReLU}(2) = 2, \qquad y = vh = 8, \qquad L = \frac{1}{2}(8 - 6)^2 = 2. \tag{S1}$$

Using the chain rule, we find that

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial h}\frac{\partial h}{\partial w}, \tag{S2}$$

and compute each term as

$$\frac{\partial L}{\partial y} = y - t = 2, \qquad \frac{\partial y}{\partial h} = v = 4, \qquad \frac{\partial h}{\partial w} = \text{ReLU}'(wx) \cdot \frac{\partial (wx)}{\partial w} = 1 \cdot x = 1, \tag{S3}$$

where we used that

$$\text{ReLU}'(z) = \begin{cases} 1 & z > 0, \\ 0 & z < 0. \end{cases} \tag{S4}$$

Therefore, we find that

$$\frac{\partial L}{\partial w} = 2 \cdot 4 \cdot 1 = 8. \tag{S5}$$

Question 4

Consider constructing a fully connected network with two hidden layers of size 128 that takes a 10×10 image with three channels as input and classifies it into five classes. All internal nodes use the ReLU activation function. The input tensor has shape `[B, 3, 10, 10]`, where `B` is the batch size, and `nn` has been imported from `torch`. Which one of the following PyTorch code snippets implements such a network correctly?

A.

```
nn.Sequential(
    nn.Linear(3, 10, 10, 128),
    nn.ReLU(),
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Linear(128, 5),
    nn.Softmax(dim=1)
)
```

B.

```
nn.Sequential(
    nn.Flatten(128),
    nn.ReLU(),
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Linear(5),
    nn.Softmax(dim=1)
)
```

C.

```
nn.Sequential(
    nn.Linear(300, 128),
    nn.ReLU(),
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Linear(128, 5),
    nn.Softmax(dim=1)
)
```

D.

```
nn.Sequential(
    nn.Flatten(),
    nn.Linear(300, 128),
    nn.ReLU(),
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Linear(128, 5),
    nn.Softmax(dim=1)
)
```

E. Don't know

Solution: We first need to flatten the input image before feeding it into a linear layer. The `Linear` module requires two non-optional arguments: the input size and the output size. The only option that both flattens the input and uses the correct dimensions is option D.

Question 5

Consider a 2D convolutional layer with no padding, a stride of 2 (both height and width), no bias, and the kernel

$$K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{3}$$

The input to the layer is

$$X = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 2 \\ 0 & 2 & 1 & 1 \end{bmatrix}. \tag{4}$$

What is the output of the layer?

A. $\begin{bmatrix} 3 & 2 \\ 0 & 3 \end{bmatrix}$

B. $\begin{bmatrix} 4 & 2 \\ 3 & 6 \end{bmatrix}$

C. $\begin{bmatrix} 1 & 0 \\ 3 & 3 \end{bmatrix}$

D. $\begin{bmatrix} 1 & 3 & 0 \\ 1 & 2 & 3 \\ 3 & 1 & 3 \end{bmatrix}$

E. Don't know

Solution: Let the output matrix be O. With stride 2, the kernel is applied at four positions:

1. Submatrix $\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}$ gives $O_{11} = 1 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 + 1 \cdot 0 = 1$.
2. Submatrix $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ gives $O_{12} = 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 = 0$.
3. Submatrix $\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ gives $O_{21} = 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 + 1 \cdot 2 = 3$.
4. Submatrix $\begin{bmatrix} 2 & 2 \\ 1 & 1 \end{bmatrix}$ gives $O_{22} = 1 \cdot 2 + 0 \cdot 2 + 0 \cdot 1 + 1 \cdot 1 = 3$.

Question 6

Consider a 2D convolutional layer with padding 1 (height and width), a stride of 2 (height and width), no bias, and 10 kernels of size `(channels_in, kernel_height, kernel_width)=(5, 3, 3)`. For an input tensor of size `(channels, height, width)=(5, 5, 5)`, what is the output size of the layer?

A. `(channels, height, width) = (10, 2, 2)`

B. `(channels, height, width) = (10, 3, 3)`

C. `(channels, height, width) = (10, 5, 5)`

D. `(channels, height, width) = (5, 3, 3)`

E. Don't know

Solution: The layer has 10 kernels, so the output has 10 channels. Padding 1 turns the 5×5 input into 7×7. With a 3×3 kernel and a stride of 2, the kernel can start at positions $0, 2, 4$ along each dimension. That gives three valid positions horizontally and three vertically. Thus, the output spatial size is 3×3, and the full output size is (10, 3, 3).

Question 7

Consider a fully connected neural network with one hidden layer of size 32 that maps an `input` tensor of size `[B, 10]` to 3 classes. The loss is computed as `loss(model(input), target)`, where `target` is a tensor of class indices of size `[B]`. Which one of the following code snippets correctly defines the network and the loss?

A.

```
model = nn.Sequential(
    nn.Linear(10, 32),
    nn.ReLU(),
    nn.Linear(32, 3)
)
loss = nn.CrossEntropyLoss()
```

B.

```
model = nn.Sequential(
    nn.Linear(10, 32),
    nn.ReLU(),
    nn.Linear(32, 3),
    nn.Softmax(dim=1)
)
loss = nn.CrossEntropyLoss()
```

C.

```
model = nn.Sequential(
    nn.Linear(10, 32),
    nn.ReLU(),
    nn.Linear(32, 3)
)
loss = nn.Softmax(dim=1)
```

D.

```
model = nn.Sequential(
    nn.Linear(10, 32),
    nn.ReLU(),
    nn.Linear(32, 1)
)
loss = nn.CrossEntropyLoss()
```

E. Don't know

Solution: The loss `nn.CrossEntropyLoss()` expects raw logits of shape `[B, 3]` and class indices of shape `[B]`. Since `CrossEntropyLoss` internally applies `log_softmax`, the network must *not* include a softmax layer. The only option that outputs logits of size `[B, 3]` and uses `CrossEntropyLoss()` correctly is option A.

Question 8

Consider the (non-scaled) dot-product attention

$$A = \text{softmax}(QK^\top)\,V,$$

where the softmax function is applied row-wise and given by

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}. \tag{5}$$

Let the query Q, key K, and value V matrices be given by

$$Q = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad K = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \qquad V = \begin{bmatrix} 2 & 0 & 6 & 0 \\ 0 & 4 & 0 & 8 \end{bmatrix}. \tag{6}$$

What is the attention output A?

A. $A = \begin{bmatrix} 1 & 2 & 4 & 6 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

B. $A = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

C. $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$

D. $A = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 2 & 4 & 6 & 8 \end{bmatrix}$

E. Don't know

Solution: First, we compute the attention logits

$$QK^\top = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \tag{S6}$$

and apply the softmax row-wise

$$\text{softmax}\left(\begin{bmatrix} 1 & 1 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix} \qquad \text{softmax}\left(\begin{bmatrix} 0 & 0 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix}. \tag{S7}$$

Thus, the attention weights are

$$W = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}. \tag{S8}$$

Multiply by V gives us

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}. \tag{S9}$$

Question 9

Consider a PyTorch implementation of a fully connected neural network with a single hidden layer of size 16 that takes an input tensor of shape `[B, 5]`. The network's output is used to parametrise the mean and standard deviation (or a parameter from which it can be computed) of a Gaussian distribution. The loss function computes the negative log probability of the target tensor `target` of shape `[B]` up to a normalisation constant. Recall that the Gaussian negative log-likelihood can be written as

$$-\log p(t \mid \mu, \sigma) = \log \sigma + \frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2 + \text{const.} \tag{7}$$

Which one of the following code snippets implements this correctly?

A.

```
model = nn.Sequential(nn.Linear(5, 16), nn.ReLU(), nn.Linear(16, 1))
def loss(y, target):
    return ((y - target)**2).mean()
```

B.

```
model = nn.Sequential(nn.Linear(5,16), nn.ReLU(), nn.Linear(16,2), nn.Softplus())
def loss(y, target):
    return (torch.log(y[:,1]) + 0.5*((target-y[:,0])/y[:,1])**2).mean()
```

C.

```
model = nn.Sequential(nn.Linear(5, 16), nn.ReLU(), nn.Linear(16, 2))
def loss(y, target):
    return (torch.log(y[:,1]) + 0.5*((target-y[:,0])/y[:,1])**2).mean()
```

D.

```
model = nn.Sequential(nn.Linear(5, 16), nn.ReLU(), nn.Linear(16, 2))
def loss(y, target):
    return (y[:,1] + 0.5*((target - y[:,0])/torch.exp(y[:,1]))**2).mean()
```

E. Don't know

Solution: The model must output both a mean and a parameter that guarantees a *positive* standard deviation. Option D outputs two values (mean and log standard deviation), constructs a positive standard deviation using `torch.exp`, and implements the Gaussian negative log-likelihood correctly (up to a constant). Therefore, option D is the correct answer.

Question 10

Consider constructing an LSTM for classifying sequences into three classes (a many-to-one mapping). Assume we have one-hot encoded an input sentence to a tensor of shape `[T, V]`, where `T` is the sequence length and `V` is the vocabulary size (there is no batch dimension). We want a PyTorch LSTM model that, given such an input, outputs unnormalised logits of shape `[3]` suitable for use with `nn.CrossEntropyLoss` together with a target tensor containing a single class index of shape `[1]`. Which one of the following code snippets defines this model correctly?

A.

```
class MyLSTM(nn.Module):
    def __init__(self, V):
        super().__init__()
        self.lstm = nn.LSTM(input_size=V, hidden_size=32)
        self.linear = nn.Linear(32, 3)

    def forward(self, x):
        x, (h_n, c_n) = self.lstm(x)
        return self.linear(x[-1])
```

B.

```
class MyLSTM(nn.Module):
    def __init__(self, V):
        super().__init__()
        self.lstm = nn.LSTM(input_size=V, hidden_size=32)
        self.linear = nn.Linear(32, 3)

    def forward(self, x):
        x, (h_n, c_n) = self.lstm(x)
        return self.linear(x[:-1])
```

C.

```python
class MyLSTM(nn.Module):
    def __init__(self, V):
        super().__init__()
        self.lstm = nn.LSTM(input_size=V, hidden_size=32)[-1]
        self.linear = nn.Linear(32, 3)

    def forward(self, x):
        x, (h_n, c_n) = self.lstm(x)
        return self.linear(x)
```

D.

```python
class MyLSTM(nn.Module):
    def __init__(self, V):
        super().__init__()
        self.lstm = nn.LSTM(input_size=V, hidden_size=32)

    def forward(self, x):
        x, (h_n, c_n) = self.lstm(x)
        return x[-1]
```

E. Don't know

Solution: For an unbatched input `x` of shape `[T, V]`, `nn.LSTM(input_size=V, hidden_size=32)(x)` returns as its first output a tensor of shape `[T, 32]` (when the input has no batch dimension). For many-to-one sequence classification, we want to use the representation of the *last* time step (shape `[32]`) and map it through a linear layer `Linear(32, 3)` to obtain logits of shape `[3]`. Option A does exactly this: it calls the LSTM, takes `x[-1]` (last time step, shape `[32]`), and passes it through `self.linear` to get a tensor of shape `[3]`.