AI Research Labs

224 questions · 44 lessons

Question *bank*

Every quiz question in the course, written as exam-style scenarios with misconception-based options. Answers and explanations are at the back.

1 Foundations of Deep Learning (20)
2 The Neuron and Linear Separability (15)
3 Multi-layer Perceptrons (20)
4 Activation Functions (24)
5 The Universal Approximation Theorem (15)
6 Loss Functions and Output Layers (20)
7 Backpropagation (30)
8 Generalisation and Data (25)
9 Regularisation (20)
10 Convolutional Neural Networks (20)
11 Network Design and PyTorch (15)



Free course with quizzes, progress tracking and Scoopy, an AI tutor:
airesearchlabs.si/learn/dl

Deep Learning Foundations · Edition November 2026 · © 2026 AI Research Labs. Free for personal study.

How to use this test. Answer all 224 questions without notes, then mark yourself with the answer key at the back. Each question has exactly one best answer. Score 75% or more and you've mastered the material; below that, reread the lessons for the questions you missed.

1.1 Machine Learning and Deep Learning

Q1 `concept` In the model $\hat{y} = mx + c$, which statement correctly separates the model from learning?

- A The line is the model; finding m and c from data is learning
- B Learning is choosing the straight-line form; the model is the fitted values
- C The model and the learning algorithm are the same thing in machine learning
- D The data points are the model, and drawing the line through them is the learning

Q2 `calculate` Learning produces $\hat{y} = 5x + 30$. What does the model predict for $x = 6$?

- A 60
- B 35
- C 180
- D 41

Q3 `concept` Which statement about machine learning (ML) and deep learning (DL) is correct?

- A DL is the part of ML that uses multi-layered neural networks
- B DL and ML are unrelated fields that happen to share some vocabulary
- C Every ML model is a neural network, and DL is just the largest ones
- D ML is the part of DL that uses models without any neural networks in them at all

Q4 `apply` Why does the learning loop include an evaluation step before going round again?

- A Because the model has to be re-chosen from scratch after every round of training
- B Because evaluation is where the loss gradient for the next round gets computed
- C Because the parameters can only be updated once the evaluation step has finished
- D Because fitting training data doesn't guarantee good predictions on new data

Q5 `interpret` Which milestone made training multi-layer networks practical, by giving a systematic way to compute their gradients?

- A Backpropagation, popularised in 1986
- B The McCulloch–Pitts neuron model of 1943
- C The ImageNet competition result of 2012
- D The ADALINE echo canceller of 1959

1.2 Types of Learning

Q6 `apply` A model predicts a student's final mark (0–100) from hours of study. What kind of task is this?

- A Supervised classification
- B Unsupervised clustering
- C Reinforcement learning
- D Supervised regression

Q7 apply You have 10,000 customer records with no labels and want to find natural groups of similar customers. Which type of learning fits?

- A Unsupervised learning
- B Reinforcement learning
- C Supervised learning
- D Semi-supervised regression with pseudo-labels

Q8 concept In the notation $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, what does the superscript (i) mean?

- A It picks out training example number i
- B It is the i-th feature of a single input vector
- C It is the layer number in the network
- D It raises x to the power i

Q9 concept What distinguishes reinforcement learning from supervised learning?

- A It always uses much deeper neural networks than supervised learning does
- B It learns from rewards for actions, not from labelled input–output pairs
- C It is simply supervised learning that is applied to time-series data only
- D It needs every training example to carry a correct numerical target label

Q10 interpret A photo app asks you to name one face, then names the same person across hundreds of other photos. Which type of learning does this illustrate?

- A Pure unsupervised learning
- B Reinforcement learning
- C Semi-supervised learning
- D Fully supervised learning

1.3 Gradient Descent and Momentum

Q11 calculate Minimise $f(w) = (w-3)^2$ by gradient descent from $w_0 = 5$ with $\eta = 0.1$. What is w_1?

- A 4.6
- B 4.8
- C 5.4
- D 3.0

Q12 interpret Your training loss shoots up and becomes NaN within a few iterations. What is the most likely cause?

- A The learning rate is too small, so training has not had enough time
- B The loss function is convex, so gradient descent cannot be used
- C The learning rate is too large, so the updates overshoot and diverge
- D The model has reached a local minimum where the gradient is zero

Q13 concept At some parameters the gradient is exactly zero. What can you conclude?

A The point is a local minimum, though perhaps not the global one

B The point is the global minimum of the objective function

C The learning rate is too small to make any further progress

D The point is stationary: a minimum, maximum, saddle point or plateau

Q14 concept What does momentum add to the gradient descent update $x_{k+1} = x_k - \eta \nabla f(x_k)$?

A A fraction of the previous step, so steady directions build speed

B A penalty on large parameters, so the learned weights stay small

C A second-derivative term, so the step size adapts to the curvature

D A random perturbation at every step, so it can escape local minima

Q15 calculate For $f(w) = (w-3)^2$, each gradient step multiplies the distance to 3 by $(1-2\eta)$. What happens with $\eta = 1$, starting at $w = 5$?

A It converges slowly towards 3

B It reaches 3 in a single step

C It moves steadily away from 3

D It bounces between 5 and 1

1.4 Notation, Probability and Jargon

Q16 calculate X is 0 with probability 0.75 and 10 with probability 0.25. What are $\mathbb{E}[X]$ and $\mathrm{Var}(X)$?

A 2.5 and 18.75

B 5.0 and 25.00

C 2.5 and 4.33

D 2.5 and 6.25

Q17 concept Which of these is a hyperparameter rather than a parameter?

A A hidden layer's weight matrix

B A single weight w_{12}

C The learning rate η

D An output unit's bias term

Q18 interpret A lecturer says a model "has high bias". In a discussion of underfitting, what do they mean?

A Its bias terms b have unusually large values

B Its model class makes systematic errors

C It treats some groups of people unfairly

D It changes a lot from one training sample to another

Q19 concept If w is a column vector (w_1, w_2, w_3), what is $w^\top x$ for $x = (x_1, x_2, x_3)$?

A The vector $(w_1 x_1, w_2 x_2, w_3 x_3)$

B A 3×3 matrix of all products

C The vector w written as a row

D The single number $w_1 x_1 + w_2 x_2 + w_3 x_3$

Q20 concept Why is the standard deviation often reported instead of the variance?

- A It is easier to differentiate than the variance
- B It is in the same units as the variable itself
- C It is always smaller than the variance is
- D It ignores the extreme values in a data set

2.1 The Artificial Neuron

Q21 calculate A neuron has $w = (0.5, 1.5)$, $b = -0.2$ and input $x = (2, -1)$. What is its pre-activation z?

- A $+2.3$
- B -0.7
- C $+0.3$
- D -0.5

Q22 calculate The same neuron (pre-activation $z = -0.7$) uses a ReLU activation. What does it output?

- A −1
- B 0
- C 0.332
- D −0.7

Q23 concept What does the bias b of a neuron do?

- A It scales every input by the same constant factor
- B It measures the systematic error of the neuron's predictions
- C It shifts the threshold, independently of the inputs
- D It is the activation function applied after the weighted sum

Q24 interpret One input of a trained neuron has weight -2.4; another has weight 0.01. What do these tell you?

- A The first input is unimportant because its weight is negative
- B Both inputs raise the output, since weights only set the speed of learning
- C The second input matters most because its weight is the smallest
- D The first input strongly lowers the output; the second barely matters

Q25 concept Why can the bias be written as an extra weight w_0?

- A Add a constant input $x_0 = 1$, so $b = w_0 \cdot 1$
- B Because the bias must be zero for gradient descent to work
- C Because the bias is always equal to the first weight
- D Because the activation absorbs the bias into its formula

2.2 Decision Boundaries and Linear Separability

Q26 concept For a neuron with a sign activation, what is its decision boundary?

- A The curve where the activation's derivative is largest
- B The set of inputs where the output is exactly 1
- C The set of inputs where $w^\top x + b = 0$
- D The line through the origin perpendicular to b

Q27 calculate A neuron with weights $(1, 1)$, bias -1.5 and a sign activation receives the four binary inputs. Which function does it compute?

- A AND
- B NAND
- C OR
- D XOR

Q28 concept Which direction does the weight vector w point relative to the decision boundary?

- A Through the origin, whatever the value of the bias
- B Along it, from one end of the boundary to the other
- C Perpendicular to it, towards the negative side
- D Perpendicular to it, towards the positive side

Q29 apply Which of these datasets is linearly separable?

- A One class inside a circle, the other in a ring around it
- B XOR on two binary inputs
- C OR on two binary inputs
- D Two interleaved spirals, one for each of the two classes

Q30 interpret You keep $w = (1, 1)$ but change the bias from -1.5 to -0.5. What happens to the boundary?

- A It becomes curved, because the bias is now smaller
- B It rotates by 45°, turning AND into XOR
- C It slides parallel to itself, turning AND into OR
- D It stays where it is, since only weights move it

2.3 The XOR Problem

Q31 concept In the proof that one neuron can't compute XOR, what contradiction do the four inequalities produce?

- A The output must be both 0.5 and 1 at the same input
- B $w_1 + w_2 + 2b$ must be both positive and negative
- C Both of the weights must be negative and also positive
- D The bias b must be both zero and also non-zero

Q32 `interpret` You train a single sigmoid neuron on XOR with MSE. The loss stalls at 0.25 and every output is about 0.5. What does this mean?

- A The learning rate is too small; train much longer and it will solve XOR
- B The training data must be mislabelled, since XOR is linearly separable
- C It has found the best a straight boundary can do: predict the mean
- D The weights have exploded, so the sigmoid outputs saturate at 0.5

Q33 `calculate` XOR can be written as $x_1 + x_2 - 2x_1x_2$. What does it give for $(1, 1)$?

- A 2
- B 0
- C 1
- D −2

Q34 `apply` You add the product x_1x_2 as a third input to a single neuron. Can it now compute XOR?

- A Only if the activation function is also changed from sign to ReLU
- B Yes: XOR is linear in (x_1, x_2, x_1x_2)
- C No: a single neuron can never compute XOR, whatever inputs it is given
- D Only with a hidden layer added after the new product feature

Q35 `concept` Why is the XOR result described as a proof rather than an experimental finding?

- A It depends on the particular learning rate that was used in training
- B It was verified by training many thousands of networks on XOR
- C It shows no weights at all can work, not just that training failed
- D It holds only for the sigmoid activation, not for any other

3.1 Layers, Matrices and Shapes

Q36 `calculate` How many trainable parameters does a 2 → 3 → 1 fully connected network have?

- A 11
- B 9
- C 6
- D 13

Q37 `calculate` How many parameters does a 100 → 50 → 10 network have?

- A 6,000
- B 5,060
- C 5,560
- D 5,500

Q38 `concept` In the lecture convention $y = f(Wx + b)$ with column vectors, a layer has 4 inputs and 6 neurons. What is the shape of W?

- A 4×6
- B 6×4
- C 4×4
- D 6×6

Q39 apply In NumPy you compute a layer as `X @ W + b`, with one example per row of `X`. A layer maps 4 inputs to 6 units. What shape must `W` have?

A 4×6
B 6×4
C 4×1
D 6×1

Q40 interpret A 784 → 128 → 10 network has 101,770 parameters. Where are almost all of them?

A In the 128 → 10 output connections
B Spread evenly between the two layers
C In the 784 → 128 connections
D In the biases of the two layers

3.2 Solving XOR with a Hidden Layer

Q41 concept In the 2–2–1 XOR network with sign units, what do the two hidden units compute?

A NAND and NOR of the two inputs
B OR and AND of the inputs
C XOR and its exact negation
D The two inputs, unchanged

Q42 calculate In that network the output is $\operatorname{sign}(h_1 - h_2 - 1)$. For input $(1, 1)$ the hidden outputs are $(+1, +1)$. What is the output?

A −3
B 0
C +1
D −1

Q43 concept Why can one output neuron separate XOR once the hidden layer has done its work?

A The hidden layer adds extra dimensions, and every dataset is separable in more dimensions
B The output neuron uses a curved decision boundary, unlike the hidden units
C The hidden layer memorises the four input patterns and looks up the answer
D The hidden layer maps the inputs into coordinates where the classes are separable

Q44 interpret A classmate writes: "the hidden layer solves XOR by mapping the data to a higher-dimensional space." Is that accurate for the 2–2–1 network?

A Yes: the hidden layer always has more units than the inputs
B Yes: separability needs a jump from 2 to 3 dimensions here
C No: it stays 2-dimensional; the non-linearity is what helps
D No: the hidden layer actually reduces the dimension to 1

Q45 apply You replace the sign units with sigmoids and train a 2–2–1 network on XOR from random starts. What should you expect?

- A It never succeeds, because sigmoids can't compute XOR
- B It usually succeeds, but some starts get stuck
- C It succeeds only if the learning rate is exactly 1
- D It always finds the exact OR/AND weights shown above

3.3 Why Depth Helps: Composing Regions

Q46 concept With threshold units, what region can a single first-layer unit describe?

- A A union of separate regions
- B A convex polygon
- C A circle around a point
- D A half-plane

Q47 concept An AND-like unit in the second layer combines several half-plane tests. What shape can it carve out?

- A A convex region, such as a triangle
- B Any non-convex shape at all, such as a ring
- C Only a single straight line through the data
- D Two disconnected regions at the same time

Q48 apply You need to classify points that fall in either of two separate triangles. How many layers of threshold units does the construction need?

- A Four: each triangle needs its own separate hidden layer
- B One: a single layer can describe any union of shapes
- C Two: the second layer can already take unions of triangles
- D Three: lines, then triangles, then their union

Q49 interpret What is the general lesson of the half-plane → convex → union construction?

- A Deeper networks always need fewer parameters than shallow ones
- B Depth matters only for classification, never for regression
- C Every layer must use the threshold activation to be useful
- D Later layers compose features detected by earlier layers

Q50 concept Why does the geometric construction come with a warning not to overstate it?

- A It is simply wrong: depth adds no expressive power over a single layer
- B It only applies to networks with exactly three layers and no more than that
- C It assumes that the data are linearly separable before the first layer
- D It uses idealised threshold units, and depth also makes training harder

3.4 Why Non-linearity Is Essential

Q51 `calculate` Two linear layers: $W_1 = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$, $b_1 = (1, 0)$, $W_2 = \begin{bmatrix} 1 & -1 \end{bmatrix}$, $b_2 = 2$. What single layer are they equivalent to?

- A $y = x_1 + 2x_2 + 2$
- B $y = x_1 + x_2 + 3$
- C $y = x_1 - x_2 + 3$
- D $y = x_1 + x_2 + 2$

Q52 `concept` What do ten stacked layers with identity activations compute?

- A A single affine function $Wx + b$
- B A piecewise-linear function with ten pieces
- C Any continuous function, given enough units
- D A polynomial of degree ten in the inputs

Q53 `apply` Can a deep network whose hidden layers all use the identity activation solve XOR?

- A Yes, if each hidden layer is wide enough
- B Yes, if it has at least two hidden layers
- C No: it collapses to one linear layer
- D Only if it is trained for long enough with a small learning rate

Q54 `interpret` Where is a linear (identity) activation the right choice?

- A At the output of a regression network
- B In every hidden layer of a deep network
- C At the output of a multi-class classifier
- D Anywhere, since all activations are equivalent

Q55 `concept` What do non-linear activations give a multi-layer network that linear ones cannot?

- A Faster training, because non-linear functions are cheaper
- B Fewer parameters, because the activations have their own weights
- C Guaranteed convergence to the global minimum of the loss
- D Layers that bend the space, so depth adds expressive power

4.1 Sigmoid, tanh, ReLU and Friends

Q56 `calculate` A sigmoid unit outputs $a = 0.8$. What is its derivative $\sigma'(z)$ at that point?

- A 0.16
- B 0.8
- C 0.2
- D 0.64

Q57 concept What is the largest value the sigmoid derivative can take, and where?

A 0.50, at $z = 0$

B 1.00, as $z \to \infty$

C 1.00, at $z = 0$

D 0.25, at $z = 0$

Q58 concept Why can't networks of sign-activation units be trained by gradient descent?

A Its derivative is zero almost everywhere, so there's no signal

B It is too expensive to compute for large networks of units

C Its derivative is infinite everywhere except exactly at zero

D Its outputs are only −1 and +1, which the loss can't measure

Q59 interpret A hidden unit has pre-activation $z = 5$. Compared with $z = 0$, how much gradient does a sigmoid pass through it?

A Far more: about 0.99 instead of 0.5

B The same, since the derivative is constant

C Exactly half as much, because the output doubled

D Far less: about 0.0066 instead of 0.25

Q60 concept What advantage does tanh have over the sigmoid for hidden units?

A It never saturates, whatever the size of the input

B Its derivative is exactly 1 everywhere it is defined

C It is zero-centred, with outputs in (−1, 1)

D It produces exact zeros for every negative input

4.2 Vanishing Gradients

Q61 calculate Backpropagating through 10 sigmoid layers multiplies 10 activation derivatives. What is the largest that product can be?

A About 0.025

B 1

C About 9.5×10^{-7}

D 2.5

Q62 interpret In a deep sigmoid network, the last layers' weights change quickly but the first layers' barely move. What explains it?

A The first layers' inputs are always zero in deep networks of this kind

B The first layers have a much smaller learning rate set by the optimiser

C The error signal shrinks at every layer it passes back through

D The loss function only depends on the weights of the last layer

Q63 apply Which change most directly reduces vanishing gradients in a deep network?

A Replace the sigmoid hidden units with ReLU
B Use a much smaller learning rate throughout
C Add a sigmoid output layer for the predictions
D Make the input features larger in magnitude

Q64 concept How does saturation make vanishing gradients worse?

A Saturation makes the learning rate negative
B Saturated units have derivatives far below 0.25
C Saturated units output exactly zero every time
D Saturated units double the error signal each time

Q65 concept In the recursion $\delta^{(k)} = D_k W_{k+1}^{\top} \delta^{(k+1)}$, what does D_k contain?

A The learning rates for each unit on its diagonal
B The weights of layer k on its diagonal
C The inputs to layer k on its diagonal
D The activation derivatives $f'(z^{(k)})$ on its diagonal

4.3 Dying ReLU, Leaky ReLU and ELU

Q66 concept What is a "dead" ReLU unit?

A One whose weights have become very large after a big update step
B One whose input is negative for every example, so it never updates
C One whose output is always exactly 1, so it passes everything through
D One that was removed from the network by dropout during training

Q67 apply Many hidden units in your ReLU network output 0 for every input after a few epochs. Which fix targets this?

A Remove the biases from all the hidden layers
B Add many more ReLU layers to the network
C Use Leaky ReLU and a smaller learning rate
D Switch every hidden unit to the sign function

Q68 calculate A Leaky ReLU with $\alpha = 0.01$ receives $z = -3$. What does it output?

A −3
B −0.03
C 0
D 0.03

Q69 concept How do vanishing gradients and dying ReLU differ?

A Vanishing affects biases, while dying affects only the weights
B Vanishing hits ReLU networks, while dying affects only sigmoid ones
C Vanishing: small factors across layers. Dying: one unit stuck at 0.
D They are the same problem, only named for different activations

Q70 interpret Why is ReLU still the default hidden activation despite dying units?

- A Because it never saturates for any input, positive or negative
- B Because it guarantees that no unit can ever stop learning at all
- C Its derivative is 1 when active, and it's cheap and sparse
- D Because its output is bounded, which keeps all activations small

4.4 Choosing Activation Functions

Q71 apply A network predicts tomorrow's temperature in °C. Which output activation fits?

- A Softmax
- B Sigmoid
- C ReLU
- D Linear

Q72 apply A network decides whether an email is spam. Which output layer fits?

- A One sigmoid unit giving $P(\text{spam})$
- B One linear unit giving an unbounded score
- C Ten softmax units, one for each spam score
- D One ReLU unit giving a non-negative score

Q73 apply A network recognises which of 10 digits is in an image. Which output layer fits?

- A 1 unit with sigmoid
- B 1 unit with linear output
- C 10 units with ReLU
- D 10 units with softmax

Q74 concept For the hidden layers of a deep network, which activation is the recommended default?

- A ReLU
- B Linear
- C Sign
- D Sigmoid

Q75 interpret Why is a sigmoid acceptable in a network with one hidden layer but discouraged in a 20-layer one?

- A Sigmoids become linear once there are more than two layers
- B Vanishing gradients only become severe with many layers
- C Sigmoids can only be used in networks with one layer
- D Deep networks require outputs bounded in (−1, 1)

Q76 apply In the 2–3–1 network from lesson 3.1, why does the output neuron use the identity activation?

- A Because output neurons have no weights
- B So the output always lies between 0 and 1
- C Because ReLU cannot be used in a last layer
- D So the prediction can be any real number

Q77 concept Is a neuron with no activation function still a neuron?

A No: without an activation it is just a weight

B No: it becomes a bias term on the next layer

C Yes: it uses the identity activation

D Yes, but only if it is in the input layer

Q78 concept A neuron applies ReLU to its weighted sum, then a sigmoid to the result. How many activation functions does it really have?

A One: the two together form a single activation

B Two, and each one needs its own weights and bias

C Two, so it counts as two separate neurons

D None, because the two functions cancel out

Q79 concept Can a neuron contain another neuron inside it?

A Yes: hidden neurons contain the input neurons

B Yes, whenever it uses more than one activation

C Only in convolutional networks, where filters nest

D No: networks are built by connecting neurons

5.1 The Universal Approximation Theorem

Q80 concept Which is a correct statement of the universal approximation theorem?

A A deep network with many layers can approximate any function, but one hidden layer can't

B A network trained by gradient descent always finds the best approximation to any function

C Any neural network with one hidden layer approximates every function exactly, on any domain

D One hidden layer with enough units can approximate any continuous function on a compact set

Q81 concept What does the "compact" in "compact set K" require of the inputs?

A They lie on a straight line in the input space

B They lie in a closed and bounded region

C They have been standardised to mean zero

D They are binary values, 0 or 1 only

Q82 concept Why does the theorem measure error with $\sup_{x \in K} |h(x) - \mathrm{NN}(x)|$?

A The bound must hold at every point of K, the worst case

B The bound only has to hold on average over all of K

C It is the derivative of the loss with respect to x

D It measures the error on the training points only

Q83 interpret Does the theorem apply to networks with ReLU activations?

A Yes: it extends to any continuous non-polynomial activation

B Only when the ReLU network has at least two hidden layers

C No: it is only proved for the sigmoid activation function

D No: ReLU networks can only represent linear functions of inputs

Q84 concept Why must the activation not be a polynomial for the theorem to hold?

A Polynomials cannot be differentiated by backpropagation
B Polynomials are too expensive to compute for large networks
C Polynomials always produce outputs that are greater than 1
D Polynomial units only build polynomials of bounded degree

5.2 Building Functions from Bumps

Q85 concept What does $\sigma(k(x-a)) - \sigma(k(x-b))$ look like for large k and $a < b$?

A A smooth S-curve from 0 to 1 centred on $(a+b)/2$
B About 1 to the left of a and 0 to the right of b
C About 1 between a and b, about 0 elsewhere
D About 2 between a and b, and 1 everywhere else

Q86 calculate Approximating a curve with 16 bumps, each built from two sigmoid units, needs how many hidden units?

A 8
B 32
C 256
D 16

Q87 interpret Going from 4 bumps to 16 bumps on the same interval mainly changes what?

A The approximation becomes worse because there are more bumps to add
B Smaller steps and lower error, at the cost of more units
C Nothing, since the theorem says 4 bumps are enough for any function
D The network becomes deeper, adding one more layer per extra bump

Q88 concept In the bump construction, where is the height of each bump set?

A By the steepness k of its two sigmoids
B By the number of hidden layers in the net
C By the output weight on that bump's units
D By the bias of the output unit on its own

Q89 apply Why is the bump construction not how a trained network actually works?

A Because real networks never use sigmoid activations
B Because bumps can only approximate linear functions
C It's a proof idea, not what gradient descent finds
D Because trained networks cannot represent bumps

5.3 What the Theorem Does Not Promise

Q90 concept Which of these does the universal approximation theorem guarantee?

A That a suitable one-hidden-layer network exists
B That the network will generalise well to new data
C That the required number of units is reasonably small
D That gradient descent will find the network's weights

Q91 interpret A 2–2–1 sigmoid network can represent XOR, yet about 20% of random starts fail to learn it. How does this relate to the theorem?

A It shows the theorem is false for small networks
B Representable is not the same as easy to train
C It shows XOR is not a continuous function at all
D It means the training data must contain errors

Q92 apply A student argues: "the theorem proves one hidden layer is enough, so deep networks are pointless." What's the best response?

A One layer may need impractically many units; depth is often far more efficient
B They're right, but only for regression; classification always needs depth
C They're right: deep networks only exist because GPUs make them faster to train
D They're wrong: the theorem actually requires at least two hidden layers to work

Q93 concept The theorem is about continuous functions. How does it still apply to classifiers with discrete labels?

A It doesn't apply: classifiers are not covered by any approximation theorem
B Discrete labels are continuous functions of the inputs, so nothing changes
C The network outputs labels directly and rounds the approximation error away
D The network approximates class probabilities, then a decision rule picks a label

Q94 interpret Does the theorem imply that a big enough network can't overfit?

A No: enough capacity to fit any function can also fit noise
B Yes: an exact approximation leaves no room for overfitting
C Yes: the theorem bounds the error on new data as well
D No, but only because the theorem needs infinite data

6.1 Training as Minimising a Loss

Q95 calculate Data $(1,2),(2,4),(3,5)$; line $\hat{y}=x$ (so $m=1$, $c=0$). What is the MSE J?

A 1.67
B 9
C 3
D 5

Q96 calculate For the same data and line, $\partial J/\partial c = -\frac{2}{N}\sum(y-mx-c)$. What is its value?

A $+10/3 \approx 3.33$
B $-5/3 \approx -1.67$
C $-10/3 \approx -3.33$
D $-22/3 \approx -7.33$

Q97 interpret At $m = 1$, $c = 0$ both gradients are negative. What will a gradient descent step do?

A Increase m but decrease c, rotating the line about its centre

B Increase both m and c, moving the line up towards the points

C Decrease both m and c, moving the line further down

D Leave m and c unchanged, since the line already passes nearby

Q98 concept What are the two decisions that define how a network is trained?

A The loss to measure error, and the optimiser to minimise it

B The number of layers, and the number of units in each one

C The activation function, and the size of the output layer

D The training set, and the test set used for final evaluation

Q99 calculate After one step with $\eta = 0.05$, the line becomes $\hat{y} = 1.367x + 0.167$ and the MSE drops to about 0.66. What was the MSE before?

A 9

B 1.5

C 3

D 0.66

6.2 MSE, MAE and Asymmetric Losses

Q100 calculate Targets $1, 2, 3, 4, 100$. Which single prediction minimises MSE, and which minimises MAE?

A MSE: 3 (the median). MAE: 22 (the mean).

B MSE: 22 (the mean). MAE: 3 (the median).

C Both are minimised by the mean, 22.

D Both are minimised by the median, 3.

Q101 concept Why does one huge outlier affect an MSE-trained model much more than an MAE-trained one?

A MSE counts each outlier twice in its average

B MSE's gradient grows with the error; MAE's stays ±1

C MAE ignores every error larger than one entirely

D MAE uses the median of the inputs, not the outputs

Q102 concept MAE is not differentiable at an error of exactly zero. How is it handled?

A A subgradient (any value in [−1, 1], usually 0) is used

B MAE is replaced by MSE whenever the error becomes zero

C Those examples are dropped from the training set entirely

D Training stops, since the minimum has then been reached

Q103 apply A shop forecasts demand. Running out costs three times as much as overstocking. Which loss fits?

A A pinball loss that weights under-prediction 3×
B Cross-entropy, because the demand is uncertain
C MAE, because it is robust to the outlying days
D MSE, because it punishes the large errors much more

Q104 apply You use a sigmoid output for a regression network predicting prices in thousands. What goes wrong?

A Nothing: sigmoid is the standard regression output
B Every prediction is trapped between 0 and 1
C The network can only learn linear functions
D The loss becomes negative for large prices

6.3 Softmax and Cross-entropy

Q105 calculate Logits $(2, 1, 0)$. What probability does softmax give the first class?

A About 0.880
B About 0.667
C About 0.333
D About 0.665

Q106 calculate The correct class receives probability 0.5. What is the cross-entropy loss (natural log)?

A About 1.386
B About 0.693
C 0.5
D About 0.301

Q107 concept What is $\partial H / \partial s_j$ for softmax followed by cross-entropy?

A $-z_j / o_j$
B $o_j(1 - o_j)$
C $(o_j - z_j)\, o_j(1 - o_j)$
D $o_j - z_j$

Q108 calculate Softmax outputs $(0.665, 0.245, 0.090)$ and the correct class is the first. What is the gradient with respect to the logits?

A $(0.665,\ 0.245,\ 0.090)$
B $(-0.335,\ 0.245,\ 0.090)$
C $(-0.335,\ 0,\ 0)$
D $(0.335,\ -0.245,\ -0.090)$

Q109 apply Your softmax overflows with logits like $(1000, 999, 998)$. What's the standard fix?

A Subtract the largest logit from all of them first

B Divide every logit by the sum of all the logits

C Use base-10 exponentials instead of base e

D Clip each logit at 1 before the exponential

6.4 Binary, Multi-class and Logits in PyTorch

Q110 calculate The true label is $y = 1$ and the model predicts $\hat{y} = 0.1$. What is the binary cross-entropy?

A About 2.303

B About 0.105

C 0.9

D About 0.693

Q111 apply Your PyTorch classifier ends with a softmax layer and you train with `nn.CrossEntropyLoss`. What's wrong?

A CrossEntropyLoss only works for two classes, so use MSELoss instead

B The loss applies softmax again; the model should return raw logits

C Nothing: CrossEntropyLoss expects probabilities, so softmax is required

D The softmax must come after the loss is computed, not before it

Q112 concept Why isn't accuracy used as the training loss?

A It is far too expensive to compute on every single mini-batch

B It always gives the same value for every possible model

C It can only be measured on the test set, not on the training data

D It's piecewise constant, so its gradient is zero almost everywhere

Q113 calculate Class A is uniform on $[0, 5]^2$, class B on $[0, 10]^2$, equal priors. What is $P(A \mid x)$ for a point inside $[0, 5]^2$?

A 0.5

B 0.8

C 1.0

D 0.25

Q114 interpret In that example the best possible accuracy is 0.875. A colleague's network reaches 0.87. What does that suggest?

A It must be overfitting, since 0.87 is suspiciously high

B It is near the Bayes limit; little more can be gained

C It is badly underfitting and needs a much larger network

D The test set must be leaking into the training data

7.1 The Chain Rule and Computational Graphs

Q115 `calculate` $s = wx$, $\hat{y} = \sigma(s)$, $E = (\hat{y} - t)^2$ with $x = 1$, $w = 0.5$, $t = 1$, giving $\hat{y} = 0.6225$. What is dE/dw?

- A About -0.235
- B About $+0.177$
- C About -0.755
- D About -0.177

Q116 `concept` A node's value feeds three different downstream nodes. How do you get the loss's derivative with respect to it?

- A Use only the path with the largest gradient
- B Average the three paths' contributions
- C Add the contributions from all three paths
- D Multiply the contributions of the three paths

Q117 `concept` What is the difference between backpropagation and gradient descent?

- A Gradient descent computes the gradient; backprop then updates the weights
- B Backprop is used for training, gradient descent only for the evaluation
- C They are two names for exactly the same algorithm in neural networks
- D Backprop computes the gradient; gradient descent uses it to update

Q118 `interpret` Why does backpropagation cost only about as much as one extra forward pass, however many weights the network has?

- A It reuses intermediate derivatives instead of recomputing them
- B It approximates each gradient with a finite-difference estimate
- C It skips every layer whose weights did not change in this step
- D It only computes the gradient for a random subset of the weights

Q119 `calculate` $f(x) = (3x + 1)^2$. Using the chain rule, what is $f'(x)$ at $x = 1$?

- A 8
- B 6
- C 16
- D 24

7.2 Backpropagation for a Single Sigmoid Neuron

Q120 `concept` For $s_i = w^\top x^{(i)} + b$, $\hat{y}_i = \sigma(s_i)$, $E = \frac{1}{N}\sum(\hat{y}_i - y_i)^2$, what is $\delta_i = \partial E / \partial s_i$?

- A $\frac{2}{N}(\hat{y}_i - y_i)$
- B $\frac{2}{N}(\hat{y}_i - y_i)\,\hat{y}_i(1 - \hat{y}_i)$
- C $(\hat{y}_i - y_i)\,x^{(i)}$
- D $\hat{y}_i(1 - \hat{y}_i)\,x^{(i)}$

Q121 concept Given the error signals δ_i, what is $\partial E/\partial w$ over a batch?

A $X^\top \delta = \sum_i \delta_i \, x^{(i)}$

B $X\delta$, the inputs times δ

C $\sum_i \delta_i$, a single number

D $\delta^\top \delta = \sum_i \delta_i^2$

Q122 calculate $x = (1, 2)$, $w = (0.1, -0.2)$, $b = 0$, target 1, $N = 1$. Then $\hat{y} = \sigma(-0.3) = 0.4256$ and $\delta = -0.2808$. What is $\partial E/\partial w$?

A $(-0.574, -1.149)$

B $(0.281, 0.562)$

C $(-0.281, -0.281)$

D $(-0.281, -0.562)$

Q123 interpret After one update with $\eta = 1$, the neuron's output rises from 0.43 to 0.80 for target 1. What does this show?

A The gradient step moved the weights in the right direction

B The learning rate must have been too large for this neuron

C The neuron has now converged and training can be stopped

D The gradient has a sign error, since the output changed

Q124 concept Why can a single sigmoid neuron trained on AND never reach a loss of exactly zero?

A AND is not linearly separable, so some error must remain

B MSE cannot be zero for any classifier with two classes

C The learning rate is too small for the loss to reach zero

D Its output reaches 0 or 1 only as $|s| \to \infty$

7.3 One Hidden Layer: Deriving Every Gradient

Q125 concept Sigmoid hidden layer, linear output y, loss $L = \frac{1}{2}(y - t)^2$. What is the output error $\delta^{(2)}$?

A $(y - t)\, a_j$

B $y - t$

C $\frac{1}{2}(y - t)$

D $(y - t)\, y(1 - y)$

Q126 concept In that network, what is the hidden error $\delta_j^{(1)}$?

A $\delta^{(2)}\, w_j^{(2)}$

B $\delta^{(2)}\, a_j^{(1)}(1 - a_j^{(1)})$

C $\delta^{(2)}\, w_j^{(2)}\, a_j^{(1)}(1 - a_j^{(1)})$

D $\delta^{(2)}\, w_j^{(2)}\, x_i$

Q127 calculate $\delta^{(2)} = -0.2689$, $w_1^{(2)} = 1$, $a_1^{(1)} = 0.7311$. What is $\delta_1^{(1)}$?

A About -0.2689

B About -0.0529

C About -0.1966

D About $+0.0529$

Q128 interpret Hidden unit 2 has output weight $w_2^{(2)} = -1$ and $\delta^{(2)} < 0$. Why is its hidden error positive?

A Hidden errors are always positive, whatever the output weights
B Unit 2 has a larger activation than unit 1, so it flips sign
C Lowering a_2 raises y, because its output weight is negative
D The sigmoid derivative of unit 2 is negative at this input

Q129 concept What general pattern do the gradients of both layers follow?

A Target minus output, multiplied by the number of hidden units
B The derivative of the activation function, summed over all units
C Learning rate × weight × input for every single connection
D Error at the weight's output end × activation at its input end

7.4 Backpropagation in Matrix Form

Q130 concept X is $N \times d$ and $\delta^{(1)}$ is $N \times h$. What is $\partial E / \partial W^{(1)}$, and its shape?

A $\delta^{(1)} X^\top$, which is $N \times N$
B $\delta^{(1)\top} X$, which is $h \times d$
C $X \delta^{(1)\top}$, which is $N \times N$
D $X^\top \delta^{(1)}$, which is $d \times h$

Q131 concept What does $\odot$ mean in $\delta^{(1)} = (\delta^{(2)} W^{(2)\top}) \odot H \odot (1 - H)$?

A Division of one matrix by the other, element by element
B Element-wise multiplication of same-shaped matrices
C The outer product of two column vectors
D Ordinary matrix multiplication of the two matrices

Q132 apply You switch the output from sigmoid + MSE to softmax + cross-entropy. Which of the four backprop steps changes?

A Only step 3, propagating back to the hidden layer
B Every step, since the loss function is different
C Only step 4, the hidden-layer weight gradients
D Only step 1, the output error $\delta^{(2)}$

Q133 calculate $\delta^{(2)}$ is $N \times k$ and $W^{(2)}$ is $h \times k$. What is the shape of $\delta^{(2)} W^{(2)\top}$?

A $N \times h$
B $N \times k$
C $h \times k$
D $k \times h$

Q134 concept How is the bias gradient $\partial E / \partial b^{(2)}$ computed from $\delta^{(2)}$?

A Average $\delta^{(2)}$ over the output units, one value per example
B Multiply $\delta^{(2)}$ by the inputs X, then sum over the examples
C Sum $\delta^{(2)}$ over the examples, one value per output unit
D Take only the first row of $\delta^{(2)}$, for the first example

7.5 Multiple Paths and Deeper Networks

Q135 concept In a network with two outputs, hidden unit z_1 feeds both. What does $\partial J / \partial w_{2,11}$ (a weight into z_1) contain?

A The product of the terms through both outputs
B Only the term through the larger of the outputs
C A sum of two terms, one through each output
D Only the term through the first output y_1

Q136 concept Where does the "sum over paths" appear in the matrix recursion $\delta^{(k)} = D_k W_{k+1}^\top \delta^{(k+1)}$?

A Nowhere: the matrix form only follows a single path
B In $W_{k+1}^\top \delta^{(k+1)}$, which sums over the units fed
C In the learning rate, which is summed over all layers
D In D_k, which adds the activation derivatives together

Q137 interpret Why does layer 1's gradient in the two-hidden-layer example "reuse the bracketed terms"?

A Those error terms were already computed for layer 2
B To approximate the gradient with less precision
C Because the bracketed terms are always equal to 1
D Because layer 1's weights are copies of layer 2's

Q138 apply You compute a gradient by hand, following one path through the network, and your gradient check disagrees. What's the likely mistake?

A Gradient checks never agree with hand derivations
B The sigmoid derivative should have been squared
C A node feeds several outputs and you missed paths
D The finite-difference step ϵ is too small

Q139 concept Unrolling $\delta^{(k)} = D_k W_{k+1}^\top D_{k+1} \cdots \delta^{(L)}$ shows what about deep networks?

A That the gradient does not depend on the weights of the later layers at all
B That the error signal grows larger at every layer it passes back through
C One activation-derivative factor per layer: the vanishing-gradient mechanism
D That deep networks have exactly as many parameters as shallow networks

7.6 Mini-batches and Gradient Checking

Q140 calculate 10,000 training examples, mini-batches of 100. How many weight updates happen in one epoch?

A 1
B 1,000,000
C 100
D 10,000

Q141 concept Why is mini-batch gradient descent the usual choice?

A It guarantees reaching the global minimum of the loss

B Frequent updates with moderate noise, and efficient on GPUs

C It needs no learning rate, unlike the other methods

D It computes the exact gradient faster than full-batch can

Q142 calculate $E(\theta) = \theta^2$ at $\theta = 3$, $\epsilon = 10^{-6}$. What does the central difference $\frac{E(\theta+\epsilon)-E(\theta-\epsilon)}{2\epsilon}$ give?

A 9

B 12

C 3

D 6

Q143 concept Why is gradient checking used as a test and not as the way to train?

A It gives the wrong sign for roughly half of the parameters

B It needs two loss evaluations per parameter, far too slow

C It only works for networks without any hidden layers

D It needs the test set, which must not be used in training

Q144 interpret Your network learns XOR, so a classmate says the hand-coded gradients must be correct. Are they right?

A Yes: any gradient error stops training completely

B No: training can succeed despite a gradient bug

C No: XOR is learned without using gradients at all

D Yes: XOR can only be learned with exact gradients

8.1 Fitting Is Not Learning

Q145 interpret Polynomials of degree 1, 4 and 15 are fitted to 20 noisy samples of $\sin 2\pi x$. Which has the lowest error on those 20 points?

A Degree 1

B Degree 15

C All three are equal

D Degree 4

Q146 concept What does "generalisation" mean for a trained network?

A Reaching the lowest possible error on the training data

B Working equally well on any kind of data whatsoever

C Using as few parameters as the problem allows

D Performing well on new data from the same distribution

Q147 interpret A large network reaches near-zero training error on a dataset whose labels were shuffled at random. What does this show?

A Training error is a reliable measure of what was learned
B It has discovered a hidden pattern in the shuffled labels
C It can memorise data that has no pattern to learn
D The labels can't have been shuffled properly after all

Q148 apply How should you decide whether your model generalises?

A Train it for more epochs and see if loss falls
B Measure its error on data it was not trained on
C Count how many parameters the network has
D Check that the training loss is close to zero

Q149 concept Between the training points, why does the degree-15 fit swing so wildly?

A High-degree polynomials can only represent periodic functions well
B It has too few parameters to follow the sine curve smoothly
C The training points were placed too close together on the axis
D It was fitted to the noise, and nothing constrains it between points

8.2 Training, Validation and Test Sets

Q150 concept What is the validation set used for?

A Choosing the architecture, learning rate and regularisation
B Computing the gradients that update the network's weights
C Storing extra copies of the training data for later use
D Giving the one final, unbiased estimate of performance

Q151 apply You try 30 configurations and report the best test-set score. What's wrong?

A Thirty configurations is too few to find a good model
B Nothing: the test set is meant for comparing models
C You should have reported the average of all 30 scores
D You tuned on the test set, so the score is optimistic

Q152 apply A medical dataset has several scans per patient. How should you split it?

A At random by scan, so every set gets a mix of scans
B By image size, so each set has similar resolutions
C By date of scan, with the oldest scans in the test set
D By patient, so each patient's scans are all in one set

Q153 concept You're forecasting next month's sales from past months. How should you split the data?

A Use k-fold cross-validation with random folds
B Train on the past and test on the most recent period
C Shuffle the months and split them at random 80/20
D Test on the oldest data and train on the newest

Q154 calculate With 5-fold cross-validation, how many times is each training example used for validation?

- A Once
- B Five times
- C Never
- D Four times

8.3 Underfitting, Overfitting and Learning Curves

Q155 interpret Training error 18%, validation error 19%, while similar models reach 5%. What's the diagnosis?

- A A good fit, since the errors are close together
- B Underfitting: both errors are high
- C Data leakage between the two sets
- D Overfitting: the validation error is higher

Q156 interpret Training error 1%, validation error 15%. What's the diagnosis?

- A The learning rate is far too low
- B Underfitting: the model is too small
- C A good fit: training error is low
- D Overfitting: there is a large gap

Q157 concept On a typical learning curve, what does the validation cost do as training continues?

- A It rises steadily from the very first epoch
- B It falls, reaches a minimum, then rises
- C It falls steadily for as long as training goes
- D It stays flat while the training cost falls

Q158 concept In the bias–variance decomposition, which term no model can reduce?

- A The model's training error
- B The noise variance σ^2
- C The model's variance term
- D The squared bias term

Q159 apply A degree-15 polynomial changes drastically when refitted on a fresh sample of 20 points. Which term is large?

- A Noise
- B Training error
- C Variance
- D Bias

8.4 Preparing the Data

Q160 `calculate` Training data for a feature range from 20 to 70. What is the min–max scaled value of 45?

- A 25
- B 0.9
- C 0.64
- D 0.5

Q161 `apply` You standardise using the mean and standard deviation of the whole dataset before splitting. What's the problem?

- A Information from the test data leaks into training
- B The mean must be computed on the test set instead
- C Nothing: scaling statistics don't affect evaluation
- D Standardisation should only be applied to the targets

Q162 `concept` Why do unscaled inputs in the thousands cause trouble for sigmoid hidden units?

- A Large inputs saturate the units and kill their gradients
- B Unscaled inputs increase the number of network parameters
- C Large inputs make the network's outputs turn negative
- D Sigmoid units cannot accept any input larger than one

Q163 `apply` A feature "colour" takes values red, green and blue. How should you feed it to the network?

- A One-hot encode it as three binary inputs
- B Standardise the colour names before training
- C Code it as 1, 2, 3 in a single numeric input
- D Drop it, since networks can only use numbers

Q164 `interpret` 99% of transactions are legitimate. Your fraud model has 99% accuracy. What should you check next?

- A The number of hidden layers in the network
- B Its recall on fraud, via the confusion matrix
- C Nothing: 99% accuracy is excellent performance
- D Its training loss, to make sure it's still falling

8.5 Hyperparameters and a Training Recipe

Q165 `interpret` The training cost oscillates wildly and occasionally jumps up. What should you change first?

- A Lower the learning rate
- B Remove the validation set
- C Add more hidden layers
- D Raise the learning rate

Q166 apply Your new network can't drive the training loss near zero even on 100 examples. What does this most likely indicate?

A The test set is too small
B The network is overfitting
C You need more training data
D A bug in the code or data

Q167 concept Why search learning rates like 0.001, 0.01, 0.1, 1 rather than 0.1, 0.2, 0.3, 0.4?

A Optimisers reject any learning rate above 0.4
B Its effect depends on the order of magnitude
C Smaller learning rates always train faster
D Learning rates must always be powers of ten

Q168 apply Training and validation errors are both high and close. Which change fits the recipe?

A Stopping training at an earlier epoch
B Testing on the test set more often
C Stronger dropout and more weight decay
D A bigger network or longer training

Q169 concept With more than two or three hyperparameters, why is random search usually preferred to grid search?

A It needs no validation set to compare runs
B It is the only method that works with Adam
C It always finds the exact global optimum
D It explores more distinct values of each one

9.1 L2 and L1 Penalties

Q170 calculate With L2 weight decay, $\eta = 0.1$ and $\lambda = 0.01$, by what factor is each weight shrunk before the usual gradient step?

A 0.001
B 0.99
C 0.9
D 0.999

Q171 concept Which penalty tends to set many weights to exactly zero?

A Neither: penalties only shrink
B Both, about equally often
C L1, the sum of absolute weights
D L2, the sum of squared weights

Q172 concept Why does L2 shrink small weights but almost never make them exactly zero?

A It is turned off once the training loss stops falling
B Its pull λw weakens as the weight gets smaller
C It adds a constant to every weight at each step
D It only applies to weights larger than one in size

Q173 apply You have 500 input features and suspect only a handful matter. Which penalty fits?

A L1, to drive the useless weights to zero
B L2 on the biases only, to shift each unit
C L2, to make every weight equal in size
D No penalty, since all features carry some signal

Q174 concept In the penalised objective $J(\theta) + \lambda\Omega(\theta)$, what happens as $\lambda \to \infty$?

A The biases are driven to zero along with the weights
B The model fits the training data perfectly
C The penalty stops having any effect on training
D All the penalised weights are driven towards zero

9.2 Early Stopping

Q175 concept In early stopping, which weights do you keep at the end?

A Those from the epoch with the lowest training cost
B Those from the very last epoch that was trained
C The average of the weights over all of the epochs
D Those from the epoch with the lowest validation cost

Q176 calculate The validation cost is lowest at epoch 31. With a patience of 15, when does training stop?

A Epoch 46
B Epoch 15
C Epoch 31
D Epoch 16

Q177 concept Why does early stopping count as regularisation?

A It adds a penalty on large weights directly into the loss
B It limits how far weights move from their small initial values
C It randomly drops units in the network during training
D It removes the hidden units that contribute least to the output

Q178 apply The validation cost ticks up for one epoch, then improves again. Why is a patience period useful here?

A It avoids stopping on a brief noisy blip
B It replaces the need for a validation set
C It increases the learning rate gradually
D It makes the training loss fall faster

Q179 `interpret` Besides preventing overfitting, what else does early stopping decide for you?

A The learning rate for each layer
B The size of each mini-batch
C The number of hidden units
D The number of training epochs

9.3 Dropout

Q180 `calculate` Inverted dropout with $p = 0.5$: activations $h = (2, 4, 1, 3)$, mask $(1, 0, 1, 0)$. What does the layer output during training?

A $(2, 0, 1, 0)$
B $(4, 8, 2, 6)$
C $(4, 0, 2, 0)$
D $(1, 0, 0.5, 0)$

Q181 `concept` In the lectured (inverted) dropout, what happens at test time?

A Activations are multiplied by $1 - p$
B Nothing is dropped and nothing is rescaled
C Units are dropped with probability p
D Only half of the units are used

Q182 `concept` What does dropout prevent, according to the usual explanation?

A Dead ReLU units that output zero for every input
B Exploding gradients caused by large learning rates
C Co-adaptation: units relying on specific other units
D Vanishing gradients in very deep sigmoid networks

Q183 `apply` Your validation accuracy looks noisy and lower than expected, and predictions change each run on the same input. What did you probably forget?

A `optimizer.zero_grad()` before evaluating
B To add dropout to the output layer too
C To raise the dropout probability p
D `model.eval()` before evaluating

Q184 `interpret` An older answer key says "each neuron is kept with probability p and activations are scaled at test time". How does it relate to the lectured version?

A It is wrong: dropout never rescales anything
B Different convention; same expected activations
C It applies only to convolutional networks
D It is the same convention in different words

9.4 Data Augmentation

Q185 apply Which augmentation would break the labels of a handwritten-digit dataset?

- A Slight zooms of a few percent
- B Small brightness changes
- C Shifts of a pixel or two
- D 180° rotations

Q186 concept When are augmentations applied?

- A Only at test time, to make predictions stable
- B Randomly to each training mini-batch only
- C Only to the validation set, to tune them
- D Once, before training, to the test set too

Q187 concept What does augmentation teach the network?

- A To reproduce the exact transformations on its output
- B To memorise every training image pixel by pixel
- C To ignore the labels and learn only from the inputs
- D Invariance to changes that shouldn't affect the label

Q188 apply Which augmentation suits an audio keyword-spotting dataset?

- A Swapping the labels of similar keywords
- B Small time shifts and added background noise
- C Reversing each recording backwards in time
- D Horizontal flips of the spectrogram image

Q189 interpret After adding augmentation, training error rises slightly but validation error falls. What does that show?

- A The validation set is now leaking into training
- B The network has become underfitted and too small
- C The regularisation is working as intended
- D The augmentation is broken and should be removed

10.1 From Fully Connected to Convolutional

Q190 calculate A 32×32×3 image is flattened and fully connected to 100 units. How many parameters does that layer have?

- A 102,400
- B 307,200
- C 3,172
- D 307,300

Q191 calculate A convolutional layer has 16 filters of size 3×3 over a 3-channel image. How many parameters does it have?

A 49,152
B 432
C 160
D 448

Q192 concept What does weight sharing mean in a convolutional layer?

A Every filter in the layer has identical weight values
B The same filter weights are used at every position
C Weights are copied between the layers of the network
D Each output unit connects to every input value

Q193 interpret Why does the conv layer's parameter count not grow when the input image gets larger?

A Its parameters are recomputed for each new image
B Convolutions ignore every pixel outside the centre
C It depends on filter size and count, not positions
D Larger images are resized down before the conv layer

Q194 concept Which two properties of images do convolutional layers exploit?

A Locality and translation of patterns
B Colour and the brightness of the pixels
C The labels and their class frequencies
D Image size and the file format used

10.2 The Convolution Operation

Q195 calculate Input size 4×4, filter 3×3, stride 1, no padding. What size is the feature map?

A 3×3
B 4×4
C 1×1
D 2×2

Q196 calculate A 3×3 patch has column sums 3, 4 and 3 (left to right). The filter weights its columns $+1, 0, -1$. What is the output?

A −6
B 10
C 6
D 0

Q197 calculate Input size 32, filter size 5, padding 2, stride 1. What is the output size?

A 30
B 28
C 32
D 16

Q198 concept What does the filter with columns $(+1, 0, -1)$ respond to most strongly?

A A single bright pixel in the centre
B A uniformly bright patch of pixels
C A horizontal edge, brighter at the top
D A vertical edge, brighter on the left

Q199 concept A layer has 16 filters applied to a 3-channel image. How many feature maps does it produce?

A 16, one per filter
B 48, filters × channels
C 3, one per channel
D 1, summed over filters

10.3 Pooling

Q200 calculate Max pooling (2×2) is applied to the window $\{1, 3, 4, 2\}$. What is the output?

A 1
B 2.5
C 4
D 10

Q201 calculate Average pooling (2×2) is applied to the window $\{0, 1, 2, 6\}$. What is the output?

A 6
B 1.5
C 2.25
D 9

Q202 calculate 2×2 pooling with stride 2 is applied to a 16×16 feature map. What size is the output?

A 15×15
B 14×14
C 4×4
D 8×8

Q203 concept How many learned parameters does a max-pooling layer have?

A One per input channel
B Four per 2×2 window
C One weight per window
D None

Q204 interpret A feature shifts by one pixel within a 2×2 pooling window. Why does max pooling's output often stay the same?

A Pooling layers retrain their weights after each shift
B The convolution before it ignores all small shifts
C The maximum within the window doesn't change
D Pooling averages the shift over the whole image

10.4 CNN Components and Feature Hierarchies

Q205 concept What is the role of the fully connected layers at the end of a CNN?

A Detect edges and corners in the raw input image
B Add the non-linearity between convolution layers
C Shrink the feature maps to reduce the computation
D Combine the extracted features into a prediction

Q206 concept In a trained CNN, what do the early layers typically detect?

A Whole objects such as faces and cars
B Edges, corners and colour blobs
C Textures and object parts like wheels
D The final class label of the image

Q207 apply Why put a ReLU after each convolution?

A To normalise the input image into the range from 0 to 1
B So stacked convolutions don't collapse into one linear map
C To share the filter weights across different positions
D To shrink the spatial size of the resulting feature maps

Q208 interpret Why does the edge → part → object hierarchy help CNNs generalise on images?

A It removes the need for any labelled training data at all
B It guarantees the same accuracy on every image dataset
C It matches how images are composed, so features are reused
D It lets the network memorise each training image exactly

Q209 concept A unit in the second 3×3 conv layer sees a 3×3 patch of layer-1 maps. How large a region of the original image does it depend on?

A 9×9
B 5×5
C 3×3
D 6×6

11.1 Depth versus Width

Q210 concept What is the main expressive advantage of adding depth rather than width?

- A Fewer parameters in every possible architecture
- B Easier optimisation with no vanishing gradients
- C Guaranteed better generalisation on any data
- D Hierarchical features that compose simpler ones

Q211 calculate Two adjacent hidden layers each have 100 units. You double both to 200. Roughly how many times more weights connect them?

- A 8 times
- B 2 times
- C 4 times
- D The same

Q212 concept What is the main practical difficulty of very deep networks?

- A They can only be used for image classification tasks
- B They are harder to optimise, e.g. vanishing gradients
- C They need far more training data than any wide network
- D They cannot represent non-linear functions at all

Q213 apply For a small tabular dataset of a few thousand rows, what depth is a sensible starting point?

- A One hidden layer with a million units in it
- B No hidden layer at all, just linear regression
- C Fifty hidden layers with residual connections
- D Two or three hidden layers

Q214 interpret Which statement balances the depth–width trade-off correctly?

- A Depth and width are interchangeable in every respect
- B Depth is efficient but harder to train; width is easy but costly
- C Width always beats depth because the theorem needs one layer
- D Depth is always better, so width should be minimal

11.2 Designing a Network for Tabular Data

Q215 apply For predicting house prices, which output layer and loss should you start with?

- A One sigmoid unit with binary cross-entropy
- B One ReLU unit with the accuracy as the loss
- C One linear unit with MSE
- D Ten softmax units, one per price band, with cross-entropy

Q216 apply A "postcode" feature has 2,000 distinct values. What's the best way to feed it in?

- A Standardise the postcode strings to have a mean of zero
- B An embedding layer with a learned vector per postcode
- C Drop the feature, since it has far too many values
- D A single numeric input coded from 1 up to 2,000

Q217 concept Why might you train on the logarithm of the price instead of the price itself?

- A It tames the skew and stops expensive houses dominating
- B Because the logarithm makes the loss function convex
- C Because networks cannot output numbers above 1,000
- D Because prices must be categorical before training

Q218 apply You have only 800 houses. Which combination best reduces the risk of overfitting?

- A A much deeper network trained for many more epochs
- B Using the test set to choose the number of epochs
- C L2 weight decay, early stopping and cross-validation
- D Removing the validation set to train on more data

Q219 interpret One feature is "final agent fee", known only after the sale completes. Should it be an input?

- A No, but only because it would need to be one-hot encoded
- B No: it isn't available when predicting, so it's leakage
- C Yes, as long as it is standardised with training statistics
- D Yes: it is strongly correlated with the price, so it helps

11.3 Neural Networks in PyTorch

Q220 concept In the standard PyTorch training loop, what is the correct order?

- A zero_grad → forward and loss → backward → step
- B step → zero_grad → forward and loss → backward
- C forward and loss → step → backward → zero_grad
- D backward → forward and loss → zero_grad → step

Q221 apply The model ends with `nn.Softmax` and you train with `nn.CrossEntropyLoss`. What should you change?

- A Replace the softmax with a sigmoid
- B Add a second softmax before the loss
- C Remove the softmax and return raw logits
- D Switch the loss function to nn.MSELoss

Q222 concept Why must `optimizer.zero_grad()` be called every iteration?

- A PyTorch adds new gradients to the old ones by default
- B It reduces the learning rate after every single step
- C It resets the model's weights back to their initial values
- D It switches the model between training and evaluation

Q223 calculate `nn.Linear(100, 50)` followed by `nn.Linear(50, 10)`. How many trainable parameters?

A 160
B 5,560
C 5,000
D 5,500

Q224 apply How do you add L2 weight decay when training with SGD in PyTorch?

A Divide the learning rate by the batch size
B Pass `weight_decay` to the optimiser
C Add `nn.Dropout` after every layer
D Call `model.eval()` before training

Answer key

Each answer comes with the reasoning, including why the most tempting wrong option is wrong.

Q1 **A** · lesson 1.1

The model is the mathematical mapping with parameters; learning is the procedure (such as gradient descent) that estimates those parameters from examples. Choosing the straight-line form is a design decision made before learning, not the learning itself.

Q2 **A** · lesson 1.1

$5 \times 6 + 30 = 60$. 35 forgets to multiply by x, 41 adds the inputs, and 180 multiplies by 30 instead of adding it.

Q3 **A** · lesson 1.1

ML is the broad field of learning model parameters from data; DL is the subset using many-layered neural networks that learn their own representations. Plenty of ML models (decision trees, linear regression) are not neural networks.

Q4 **D** · lesson 1.1

Evaluation checks the model on data it hasn't learned from, which is the real goal (Section 8). It isn't where gradients come from (that's the learning step), and the loop refines the model rather than restarting it.

Q5 **A** · lesson 1.1

Backpropagation computes the gradient of the loss for every weight efficiently, which is what multi-layer training needs. The 1943 model was an abstraction of one neuron, ADALINE was a single-layer application, and 2012 came after backpropagation had long been in use.

Q6 **D** · lesson 1.2

There are labelled examples (hours, mark), so it's supervised, and the target is a number, so it's regression. Classification would predict a category such as pass/fail; clustering has no labels.

Q7 **A** · lesson 1.2

With no labels, the goal is to discover structure: unsupervised learning (clustering). Supervised learning needs target labels, and reinforcement learning needs an agent receiving rewards from actions.

Q8 **A** · lesson 1.2

The bracketed superscript indexes the training example. A power would be written without brackets, a feature is usually a subscript (x_j), and layer numbers use their own notation. Mixing these up is a classic source of errors.

Q9 **B** · lesson 1.2

A reinforcement learner acts in an environment and receives rewards or penalties, learning a policy. Supervised learning needs correct labels. Network depth and data type don't define either one.

Q10 **C** · lesson 1.2

A few labels (the one face you named) are combined with lots of unlabelled data (the clustered faces): semi-supervised learning. It isn't fully supervised, since most photos were never labelled, and no rewards are involved.

Q11 **A** · lesson 1.3

$f'(w) = 2(w - 3) = 4$ at $w = 5$, so $w_1 = 5 - 0.1 \times 4 = 4.6$. Adding the gradient (5.4) moves uphill; 4.8 forgets the factor 2; jumping straight to 3 would need $\eta = 0.5$.

Q12 **C** · lesson 1.3

An exploding loss is the signature of a learning rate that overshoots the minimum by more each step. A too-small rate makes progress slow but steady, a local minimum makes the loss flatten, and convexity helps rather than hurts.

Q13 **D** · lesson 1.3

A zero gradient is necessary for an interior minimum but not sufficient: maxima, saddle points and flat plateaus also have zero gradient. It says nothing about the learning rate.

Q14 **A** · lesson 1.3

Momentum adds $\alpha \Delta x_k = \alpha(x_k - x_{k-1})$: steps that agree accumulate, zig-zags cancel. Random noise, second derivatives and weight penalties are different ideas (noise injection, Newton-type methods, regularisation).

Q15 **D** · lesson 1.3

$1 - 2\eta = -1$, so $w - 3$ flips sign but keeps size 2: $5 \to 1 \to 5 \to \ldots$. One-step convergence needs $\eta = 0.5$ (factor 0), steady convergence needs $0 < \eta < 0.5$, and divergence needs $\eta > 1$.

Q16 **A** · lesson 1.4

$\mathbb{E}[X] = 0(0.75) + 10(0.25) = 2.5$. $\mathrm{Var}(X) = 0.75(0 - 2.5)^2 + 0.25(10 - 2.5)^2 = 4.6875 + 14.0625 = 18.75$. 5.0 and 25.00 treat both values as equally likely, 6.25 is only $(\mathbb{E}[X])^2$, and 4.33 is the standard deviation $\sqrt{18.75}$, not the variance.

Q17 **C** · lesson 1.4

Hyperparameters are set outside gradient-based learning (learning rate, layer sizes, dropout rate) and chosen on validation data. Weights and biases are parameters learned by gradient descent.

Q18 **B** · lesson 1.4

In the bias–variance sense, bias is systematic error from a too-simple model: underfitting. The word "bias" also names a neuron's b and unfair treatment of groups, so context decides; here it's about underfitting. High variance is the opposite problem.

Q19 **D** · lesson 1.4

$w^\top x$ is a row times a column: a dot product, one number. The element-wise product would be a vector, $x w^\top$ (column times row) would be a 3×3 matrix, and $w^\top$ alone is the row vector.

Q20 **B** · lesson 1.4

Variance is in squared units; its square root, the standard deviation, is in the original units, so it's easier to interpret. It isn't always smaller (a variance below 1 has a larger square root), and it is just as sensitive to extremes.

Q21 **B** · lesson 2.1

$z = 0.5 \times 2 + 1.5 \times (-1) - 0.2 = 1 - 1.5 - 0.2 = -0.7$. $+0.3$ adds the bias with the wrong sign, -0.5 forgets it, and 2.3 ignores the negative input.

Q22 **B** · lesson 2.1

ReLU is $\max(0, z)$, and z is negative, so the output is 0. -0.7 would be a linear activation, 0.332 the sigmoid of -0.7, and -1 the sign function.

Q23 **C** · lesson 2.1

The bias is added to the weighted sum, moving where the neuron switches on; equivalently it's a weight on a constant input of 1. It doesn't scale inputs, it isn't the "bias" of bias–variance, and it isn't the activation.

Q24 **D** · lesson 2.1

A large negative weight means more of that input pushes the pre-activation down strongly. A weight near zero means the input has almost no influence. The sign gives direction and the size gives strength.

Q25 **A** · lesson 2.1

With an extra input fixed at 1, the term $w_0 x_0 = w_0$ plays the role of the bias, and the neuron becomes $f\left(\sum_{i=0}^{n} w_i x_i\right)$. The bias isn't tied to the first weight, needn't be zero, and isn't part of the activation.

Q26 **C** · lesson 2.2

The neuron switches between -1 and $+1$ where its pre-activation changes sign, i.e. on the hyperplane $w^\top x + b = 0$. The bias moves it off the origin, and w (not b) is perpendicular to it.

Q27 **A** · lesson 2.2

$x_1 + x_2 - 1.5$ is positive only for $(1, 1)$, where it equals 0.5. That's AND. With bias -0.5 it would be OR; XOR can't be computed by one neuron; NAND is the negation of AND.

Q28 **D** · lesson 2.2

For two points on the boundary, $w^\top (x_a - x_b) = 0$, so w is normal to every direction along it. Moving along w increases z, so w points towards the side classified positive.

Q29 **C** · lesson 2.2

A single line separates $(0, 0)$ from the three OR-positive points. XOR, a circle inside a ring, and interleaved spirals all need curved or multi-part boundaries.

Q30 **C** · lesson 2.2

The weights fix the boundary's orientation; the bias only moves it. The line $x_1 + x_2 = 1.5$ shifts to $x_1 + x_2 = 0.5$, which now puts $(1, 0)$ and $(0, 1)$ on the positive side: OR. A linear neuron's boundary is always straight.

Q31 **B** · lesson 2.3

The positive cases give $w_1 + b > 0$ and $w_2 + b > 0$, which add to $w_1 + w_2 + 2b > 0$. The negative cases give $b < 0$ and $w_1 + w_2 + b < 0$, which add to $w_1 + w_2 + 2b < 0$. Impossible.

Q32 **C** · lesson 2.3

No straight boundary separates XOR, so the MSE-optimal answer is the average target 0.5 everywhere, with loss $\frac{1}{4}\sum(0.5 - y)^2 = 0.25$. More training can't help, saturated sigmoids output 0 or 1 (not 0.5), and XOR is not linearly separable.

Q33 **B** · lesson 2.3

$1 + 1 - 2(1)(1) = 0$, matching XOR(1, 1) = 0. The $-2x_1x_2$ term cancels the double count when both inputs are on; without it you'd get 2.

Q34 **B** · lesson 2.3

With weights $(1, 1, -2)$ on (x_1, x_2, x_1x_2) and a suitable bias, one neuron separates XOR. The impossibility proof is for the original two inputs. The catch is that someone had to design the feature; a hidden layer learns such features.

Q35 **C** · lesson 2.3

The inequalities rule out every possible w_1, w_2, b, so no training method or setting can succeed. A failed experiment could be bad luck or a bad learning rate; a contradiction can't. It holds for any linear decision boundary.

Q36 **D** · lesson 3.1

Layer 1: 3×2 weights $+3$ biases $= 9$. Layer 2: $1 \times 3 + 1 = 4$. Total 13. 9 forgets the second layer, 6 counts only layer-1 weights, and 11 drops a layer's biases.

Q37 **C** · lesson 3.1

$100 \times 50 + 50 = 5{,}050$ and $50 \times 10 + 10 = 510$, total 5,560. 5,500 leaves out all biases, 5,060 drops the second layer's weights, and 6,000 multiplies the widths wrongly.

Q38 **B** · lesson 3.1

Each of the 6 neurons has a row of 4 weights, so W is outputs × inputs, 6×4, and Wx with x of length 4 gives 6 values. 4×6 is the code convention (XW), used with one example per row.

Q39 **A** · lesson 3.1

X is $N \times 4$; for XW to give $N \times 6$, W must be 4×6 (inputs × outputs), the transpose of the lecture's convention. Mixing the two conventions is a common source of shape errors.

Q40 **C** · lesson 3.1

$784 \times 128 = 100{,}352$ weights sit in the first layer; the second has only $1{,}290$ parameters. Parameters concentrate where wide layers meet, which is why flattened images need so many weights.

Q41 **B** · lesson 3.2

Unit 1 computes $x_1 + x_2 - 0.5$ (OR) and unit 2 computes $x_1 + x_2 - 1.5$ (AND). The output $h_1 - h_2 - 1$ is then "OR and not AND", which is XOR.

Q42 **D** · lesson 3.2

$1 - 1 - 1 = -1$, so the sign is -1: XOR(1,1) is false. $+1$ would be wrong for (1,1); 0 is the pre-activation of a different input; -3 would come from subtracting all terms.

Q43 **D** · lesson 3.2

The non-linear hidden mapping sends (1,0) and (0,1) to the same point and moves (0,0) and (1,1) apart, so a single line separates them. Here the hidden layer has the same dimension (2) as the input, so extra dimensions aren't the reason.

Q44 **C** · lesson 3.2

The hidden layer has 2 units, as many as the inputs. What makes the classes separable is the non-linear (sign) mapping, which folds the space. More dimensions can help in other settings, but they aren't needed here.

Q45 **B** · lesson 3.2

Sigmoid networks can represent XOR, and gradient descent finds a solution in most runs, but with only 2 hidden units a share of starts get stuck (units become identical or saturate). It finds some solution, rarely the hand-built one, and no single magic learning rate is required.

Q46 **D** · lesson 3.3

One unit tests which side of a line a point is on: a half-plane. Intersections of half-planes (convex polygons) need a second, AND-like layer; unions need a third, OR-like layer.

Q47 **A** · lesson 3.3

"All of these half-planes" is their intersection, which is always convex. Non-convex or disconnected regions need a further OR-like layer that takes the union of several convex pieces.

Q48 **D** · lesson 3.3

Layer 1 draws the six lines, layer 2 forms each triangle (AND of three lines), and layer 3 joins the two triangles (OR). That's the compositional picture of what each extra layer adds.

Q49 **D** · lesson 3.3

Depth lets a network build complex functions from simpler pieces, like edges → parts → objects in images. It doesn't guarantee fewer parameters in every case, doesn't require threshold units, and applies to regression too.

Q50 **D** · lesson 3.3

The picture assumes sign units, which can't be trained by gradients, and real deep networks face problems such as vanishing gradients. The construction itself is correct and general; it just isn't the whole story.

Q51 **B** · lesson 3.4

$W_2W_1 = [1 \cdot 1 + (-1) \cdot 0,\ 1 \cdot 2 + (-1) \cdot 1] = [1, 1]$ and $W_2b_1 + b_2 = 1 + 2 = 3$. Forgetting to push b_1 through W_2 gives a bias of 2; using only W_1's first row gives $x_1 + 2x_2$.

Q52 **A** · lesson 3.4

Each layer is affine, and composing affine maps gives another affine map: $W_2(W_1x + b_1) + b_2 = (W_2W_1)x + (W_2b_1 + b_2)$, and so on. Depth adds nothing without non-linear activations.

Q53 **C** · lesson 3.4

However many linear layers you stack, the result is one affine map whose decision boundary is a hyperplane, and XOR is not linearly separable. Width, depth and training time can't change that.

Q54 **A** · lesson 3.4

A regression output must be able to take any real value, so a linear output fits. In hidden layers it would make the layers collapse; a classifier needs softmax or sigmoid outputs to produce probabilities.

Q55 **D** · lesson 3.4

Between layers, a non-linearity stops the composition from simplifying to one affine map, so each layer can transform the representation (as the XOR hidden layer did). It doesn't make training cheaper, add parameters, or guarantee convergence.

Q56 **A** · lesson 4.1

Using the output directly, $\sigma' = a(1 - a) = 0.8 \times 0.2 = 0.16$. The tempting 0.8 is the output itself, 0.2 is only $1 - a$, and 0.64 is a^2.

Q57 **D** · lesson 4.1

$\sigma'(z) = \sigma(1 - \sigma)$ peaks when $\sigma = 0.5$, i.e. $z = 0$, giving $0.5 \times 0.5 = 0.25$. A derivative of 1 at 0 belongs to tanh; 0.5 is the sigmoid's value there, not its slope.

Q58 **A** · lesson 4.1

Small weight changes don't change a sign unit's output unless they cross zero, so the derivative is 0 (and undefined at 0). Gradient descent gets no direction to move in. The loss can measure ±1 outputs fine, and the function is cheap.

Q59 **D** · lesson 4.1

At $z = 5$, $\sigma \approx 0.993$ and $\sigma' = 0.993 \times 0.007 \approx 0.0066$, almost 40 times smaller than 0.25. The unit is saturated. 0.99 is the output, not the derivative.

Q60 **C** · lesson 4.1

tanh is a rescaled sigmoid ($2\sigma(2z) - 1$) centred on zero, which often helps training. It still saturates for large $|z|$, its derivative $1 - \tanh^2$ is only 1 at zero, and exact zeros are a ReLU property.

Q61 **C** · lesson 4.2

Each factor is at most 0.25, so the product is at most $0.25^{10} = 4^{-10} \approx 9.5 \times 10^{-7}$. 2.5 adds the factors, 0.025 multiplies 0.25 by 0.1, and 1 would need every derivative to be 1 (as for active ReLUs).

Q62 **C** · lesson 4.2

Each layer multiplies the backpropagated signal by $\sigma' \leq 0.25$ (and the weights), so early layers receive tiny gradients: vanishing gradients. The learning rate is shared, inputs aren't zero, and the loss depends on every weight.

Q63 **A** · lesson 4.2

ReLU's derivative is 1 for active units, so it doesn't shrink the signal at each layer. Larger inputs push sigmoids into saturation (worse), and the learning rate scales updates without fixing the vanishing signal.

Q64 **B** · lesson 4.2

For large $|z|$ the sigmoid is nearly flat, so σ' is close to 0, much smaller than its maximum 0.25. Saturated sigmoids output values near 0 or 1, not exactly zero, and they shrink rather than amplify the signal.

Q65 **D** · lesson 4.2

$D_k = \mathrm{diag}(f'(z^{(k)}))$ multiplies the error by each unit's activation derivative. Unrolling the recursion gives one such factor per layer, which is the vanishing-gradient mechanism. Weights appear separately as $W^\top$.

Q66 **B** · lesson 4.3

If $z < 0$ for all training examples, the unit outputs 0 and its gradient is 0, so its incoming weights never change and it stays inactive. Dropout removes units only temporarily, and ReLU outputs aren't capped at 1.

Q67 **C** · lesson 4.3

Leaky ReLU keeps a small slope for negative inputs, so dead units can recover, and a smaller learning rate avoids the big steps that kill them. Sign units can't be trained at all, and more ReLU layers can add more dead units.

Q68 **B** · lesson 4.3

For $z < 0$ the output is $\alpha z = 0.01 \times (-3) = -0.03$. Plain ReLU would give 0; -3 would be the identity; the sign stays negative.

Q69 **C** · lesson 4.3

Vanishing gradients come from multiplying many derivatives below 1 (typical of sigmoids), so early layers learn slowly. Dying ReLU is a single unit with zero output and zero gradient. Different mechanisms, different remedies.

Q70 **C** · lesson 4.3

Active ReLUs pass gradients without shrinking them, the function is very cheap, and many units output exactly 0 (sparsity). It does "saturate" at 0 for negative inputs (that's dying), and its output is unbounded above.

Q71 **D** · lesson 4.4

A temperature can be any real number, including negative values, so the output must be unbounded: linear. A sigmoid caps it in (0, 1), softmax is for class probabilities, and ReLU can't output negative temperatures.

Q72 **A** · lesson 4.4

Binary classification needs one probability, which a sigmoid provides. A linear unit isn't a probability, ten softmax units would be a 10-class problem, and ReLU outputs aren't bounded by 1.

Q73 **D** · lesson 4.4

Softmax over 10 units gives one probability per digit, summing to 1. A single sigmoid handles only two classes, ReLU outputs don't form a distribution, and a single linear number would impose a false order on the digits.

Q74 **A** · lesson 4.4

ReLU avoids shrinking gradients for active units and is cheap. Sign can't be trained, sigmoid causes vanishing gradients in deep stacks (fine for shallow nets), and linear hidden layers collapse into one.

Q75 **B** · lesson 4.4

One or two factors of at most 0.25 are tolerable; twenty of them leave the early layers with almost no gradient. Sigmoids work at any depth in principle and stay non-linear; the issue is purely the gradient signal.

Q76 **D** · lesson 4.4

The network does regression, so its output must be free to take any value. ReLU can be used in a last layer, but it would forbid negative predictions; a range of 0 to 1 is what a sigmoid gives; and the output neuron has three weights and a bias of its own.

Q77 **C** · lesson 4.4

"No activation" means the identity, $f(z) = z$: the neuron still computes a weighted sum plus a bias and passes it on. Regression output neurons are like this, and linear regression is exactly one such neuron. Input-layer units hold the data and compute nothing, so they aren't the example.

Q78 **A** · lesson 4.4

A neuron computes one weighted sum and applies one function to it. Applying two functions in a row is just one composite function, $\sigma(\mathrm{ReLU}(z))$, with no extra weights. It doesn't become two neurons, and the functions don't cancel: the result lies between 0.5 and 1.

Q79 **D** · lesson 4.4

A neuron is the smallest unit: weights, a bias and one activation. Structure comes from connections, with each layer's outputs feeding the next layer. Hidden neurons receive the inputs; they don't contain them. Convolutional filters are shared weights, not neurons inside neurons.

Q80 **D** · lesson 5.1

The theorem: for a continuous target on a closed, bounded set and any tolerance ε, some one-hidden-layer network with finitely many units is within ε everywhere. It's approximate (not exact), needs a compact domain, says one layer suffices, and says nothing about training.

Q81 **B** · lesson 5.1

A compact subset of $\mathbb{R}^d$ is closed and bounded, such as $[0, 1]^d$. The guarantee is only about inputs in that region; it says nothing about extrapolating outside it. Binary or standardised inputs aren't what compactness means.

Q82 **A** · lesson 5.1

The supremum is the largest error anywhere on K, so keeping it below ε means the approximation is good everywhere at once (uniform approximation). An average-error bound would allow large errors on small regions.

Q83 **A** · lesson 5.1

Cybenko's original proof used sigmoids; Leshno and colleagues (1993) extended it to every continuous activation that isn't a polynomial, which includes ReLU. ReLU networks are piecewise linear, not linear.

Q84 **D** · lesson 5.1

If each unit is a polynomial of degree d, a one-hidden-layer network is a polynomial of degree at most d, and such a fixed-degree family can't approximate every continuous function. Polynomials are easy to differentiate and cheap; that isn't the issue.

Q85 **C** · lesson 5.2

The first steep sigmoid switches on at a, the second at b; subtracting leaves a "bump" of height about 1 on $[a, b]$. It's the difference of two steps, not a single S-curve.

Q86 **B** · lesson 5.2

Each bump uses two hidden units (one step up, one step down), so 16 bumps need 32. 16 counts bumps, not units; 256 squares the number of bumps.

Q87 **B** · lesson 5.2

Narrower intervals mean a finer staircase that tracks the target more closely, but each extra bump costs two more hidden units. The bumps all sit in the same single hidden layer; depth doesn't change.

Q88 **C** · lesson 5.2

The output unit sums the bumps weighted by its incoming weights, so each weight scales one bump to match the target's height in that interval. k controls how sharp the edges are, not the height.

Q89 **C** · lesson 5.2

The construction shows a one-hidden-layer network *can* represent any continuous function. Training finds whatever weights reduce the loss, usually much more distributed solutions. Sigmoids are used, bumps can approximate non-linear functions, and networks can represent bumps.

Q90 **A** · lesson 5.3

It is purely an existence result. It gives no training method or convergence guarantee, no practical bound on size, and nothing about generalisation or how much data is needed.

Q91 **B** · lesson 5.3

The theorem (and the explicit XOR construction) say suitable weights exist; they say nothing about whether gradient descent finds them from a given start. Failed runs get stuck with identical or saturated hidden units. Width 3+ succeeded every time.

Q92 **A** · lesson 5.3

The theorem allows an enormous number of units. Deep networks compose features and can represent many functions with far fewer units, and often train and generalise better on structured data. The theorem itself needs only one hidden layer, for regression or classification.

Q93 **D** · lesson 5.3

A classifier outputs probabilities through a sigmoid or softmax, which are continuous functions of the input that the network can approximate. A decision rule (threshold or argmax) then turns them into labels. Labels themselves jump, so they aren't continuous.

Q94 **A** · lesson 5.3

The theorem is about representing a function, not learning one from a finite noisy sample. A network flexible enough to approximate anything can memorise noise, which is exactly overfitting. The theorem says nothing about data or generalisation.

Q95 **C** · lesson 6.1

Residuals $y - \hat{y}$ are $1, 2, 2$; squares $1, 4, 4$; mean $9/3 = 3$. 9 is the sum, not the mean; 1.67 averages the residuals instead of their squares.

Q96 **C** · lesson 6.1

The residuals sum to $1 + 2 + 2 = 5$, so $-\frac{2}{3} \times 5 = -3.33$. Dropping the 2 gives -1.67; -7.33 is $\partial J/\partial m$, which weights each residual by x.

Q97 **B** · lesson 6.1

The update subtracts the gradient: $\theta \leftarrow \theta - \eta \nabla J$, so negative gradients increase the parameters. The points lie above the line (positive residuals), so moving up is right.

Q98 **A** · lesson 6.1

Training means choosing a loss $J(\theta)$ that captures the task's error, then minimising it, usually with gradient descent. Architecture and data splits matter, but they're separate decisions from how training works.

Q99 **C** · lesson 6.1

The starting line $\hat{y} = x$ had MSE 3 (squared residuals $1, 4, 4$, mean 3). One gradient step cut it to about 0.66. 9 is the unaveraged sum of squares.

Q100 **B** · lesson 6.2

Squared error is minimised by the mean, $110/5 = 22$; absolute error by the median, 3. The single outlier (100) drags the MSE answer far from the typical value, which is why MAE is called robust.

Q101 **B** · lesson 6.2

The derivative of e^2 is $2e$, so a residual of 100 pulls 100 times harder than a residual of 1. The derivative of $|e|$ is ± 1, the same size for every error. MAE doesn't ignore large errors; it just doesn't amplify them.

Q102 **A** · lesson 6.2

At zero, $|e|$ has a corner; any slope between −1 and 1 is a valid subgradient, and frameworks use 0. Examples aren't dropped, the loss isn't swapped, and one zero error doesn't mean the overall minimum.

Q103 **A** · lesson 6.2

The pinball loss charges αr for over-prediction and $-\beta r$ for under-prediction; with $\beta = 3\alpha$ the model learns to aim high. MSE and MAE are symmetric, and cross-entropy is for classification.

Q104 **B** · lesson 6.2

A sigmoid outputs values in (0, 1), so prices in the hundreds are unreachable (and the unit saturates). Use a linear output for regression. The loss stays non-negative and the hidden layers are still non-linear.

Q105 **D** · lesson 6.3

$e^2/(e^2 + e^1 + e^0) = 7.389/11.107 \approx 0.665$. 0.667 is $2/3$, i.e. normalising the raw logits without exponentiating; 0.880 is $\sigma(2)$, the sigmoid of one logit.

Q106 **B** · lesson 6.3

$-\ln 0.5 \approx 0.693$. 0.301 uses base-10 logs, 1.386 is $-\ln 0.25$, and 0.5 is the probability itself.

Q107 **D** · lesson 6.3

The softmax derivative cancels the $1/o_j$ from the log, leaving predicted minus target. $o_j(1 - o_j)$ is only part of the softmax derivative, the third option wrongly adds an activation factor, and $-z_j/o_j$ is the derivative with respect to o_j, not s_j.

Q108 **B** · lesson 6.3

$o - z$ with $z = (1, 0, 0)$ gives $(-0.335, 0.245, 0.090)$: push the correct logit up, the others down. The wrong classes still get non-zero gradients, and the sign convention is predicted minus target.

Q109 **A** · lesson 6.3

Subtracting the maximum gives $(0, -1, -2)$; the common factor cancels in the ratio, so the probabilities are unchanged and nothing overflows. Dividing or clipping changes the probabilities; base 10 overflows too.

Q110 **A** · lesson 6.4

For $y = 1$ the loss is $-\ln \hat{y} = -\ln 0.1 \approx 2.303$: a confident mistake. 0.105 is $-\ln 0.9$, what you'd get if y were 0; 0.693 is $-\ln 0.5$.

Q111 **B** · lesson 6.4

nn.CrossEntropyLoss applies log-softmax internally, so a softmax in the model gets applied twice: training runs but the gradients are distorted. Return logits and apply softmax only for reporting probabilities. It handles any number of classes.

Q112 **D** · lesson 6.4

Accuracy only changes when a prediction crosses the threshold, so small weight changes leave it unchanged: no gradient to follow. It's cheap and can be measured on any set; train on cross-entropy and report accuracy.

Q113 **B** · lesson 6.4

Densities there are $1/25$ for A and $1/100$ for B: $P(A \mid x) = \frac{1/25}{1/25+1/100} = 0.8$. 0.5 ignores the densities, 0.25 is the area ratio of the squares, and 1.0 ignores that B also covers the small square.

Q114 **B** · lesson 6.4

Where the classes overlap, no classifier can beat the Bayes-optimal accuracy, 0.875 here. A model at 0.87 is essentially there; a bigger network can't push past the ceiling, and nothing about 0.87 suggests leakage.

Q115 **D** · lesson 7.1

Multiply the local derivatives: $2(\hat{y} - t) = -0.755$, $\hat{y}(1 - \hat{y}) = 0.235$, $ds/dw = x = 1$, giving -0.177. -0.755 stops after the first factor, and the sign is negative because the output is below the target.

Q116 **C** · lesson 7.1

Multivariate chain rule: the total derivative is the sum over every path from the node to the loss. Along a single path you multiply; across paths you add. Ignoring or averaging paths gives the wrong gradient.

Q117 **D** · lesson 7.1

Backpropagation (reverse-mode automatic differentiation) calculates $\partial L/\partial \theta$ for every parameter; gradient descent then takes the step $\theta \leftarrow \theta - \eta \nabla L$. In PyTorch: `loss.backward()` then `optimizer.step()`.

Q118 **A** · lesson 7.1

Going backwards from the loss, each layer's error signal is computed once and reused for all the weights below it. Finite differences would need two evaluations per weight; backprop computes all gradients exactly, not a sample.

Q119 **D** · lesson 7.1

Outer derivative $2(3x + 1) = 8$ at $x = 1$; inner derivative 3; product 24. Stopping at the outer derivative gives 8, and $2 \times 3 = 6$ forgets to evaluate the inner expression.

Q120 **B** · lesson 7.2

Chain rule through $\hat{y}$: the loss derivative $\frac{2}{N}(\hat{y}_i - y_i)$ times the sigmoid derivative $\hat{y}_i(1 - \hat{y}_i)$. Without the sigmoid factor you'd have a linear output; including $x^{(i)}$ already gives the weight gradient, not δ.

Q121 **A** · lesson 7.2

$\partial s_i/\partial w = x^{(i)}$, and every example contributes, so the gradient is $\sum_i \delta_i x^{(i)} = X^\top \delta$, a vector shaped like w. $\sum_i \delta_i$ is the bias gradient.

Q122 **D** · lesson 7.2

$\partial E/\partial w = \delta x = -0.2808 \times (1, 2)$. Forgetting x gives the bias gradient twice; the sign is negative because $\hat{y}$ is below the target; $(-0.574, -1.149)$ uses $(\hat{y} - y)$ without the factors 2 and $\hat{y}(1 - \hat{y})$.

Q123 **A** · lesson 7.2

Moving against the gradient reduced the error: the output moved towards the target. That's the expected behaviour, not a sign error, and 0.80 is still short of 1, so training would continue.

Q124 **D** · lesson 7.2

The sigmoid approaches 0 and 1 asymptotically, so exact targets need infinite weights; the loss shrinks forever without hitting 0. AND is linearly separable, so the error does go to almost nothing.

Q125 **B** · lesson 7.3

$\partial L/\partial y = y - t$; the $\frac{1}{2}$ cancels the 2 from the square, and a linear output adds no activation-derivative factor. The $y(1-y)$ factor would apply to a sigmoid output; multiplying by a_j gives an output-weight gradient.

Q126 **C** · lesson 7.3

Chain rule: $\partial L/\partial y = \delta^{(2)}$, $\partial y/\partial a_j = w_j^{(2)}$, and $\partial a_j/\partial z_j = a_j(1-a_j)$. Dropping the output weight or the sigmoid derivative breaks the chain; multiplying by x_i gives a weight gradient, not the error.

Q127 **B** · lesson 7.3

$-0.2689 \times 1 \times 0.7311 \times 0.2689 = -0.0529$. -0.1966 is $\delta^{(2)} a_1$ (the output-weight gradient), -0.2689 drops the sigmoid derivative, and the sign stays negative because $w_1^{(2)} > 0$.

Q128 **C** · lesson 7.3

$\delta_2^{(1)} = \delta^{(2)} w_2^{(2)} a_2(1-a_2)$: two negatives make a positive. Since a_2 feeds the output with weight -1, decreasing it increases y towards the target. The sigmoid derivative is always positive.

Q129 **D** · lesson 7.3

$\partial L/\partial w_j^{(2)} = \delta^{(2)} a_j^{(1)}$ and $\partial L/\partial w_{ji}^{(1)} = \delta_j^{(1)} x_i$: the error signal where the weight delivers, times the value it carries. The learning rate belongs to the update, not the gradient.

Q130 **D** · lesson 7.4

Weight gradient = (layer input)$^\intercal$ × (error at the layer's output): $X^\top \delta^{(1)}$, $d \times h$, the same shape as $W^{(1)}$ in the code convention. $\delta^\top X$ has the transposed shape $h \times d$, which matches the lecture's W, not this one.

Q131 **B** · lesson 7.4

$\odot$ (the Hadamard product) multiplies matching entries. Each unit's backpropagated error is scaled by its own sigmoid derivative $h(1-h)$, which is an element-wise operation, not a matrix product.

Q132 **D** · lesson 7.4

The loss only enters through $\delta^{(2)}$, which becomes $(O - Z)/N$. Steps 2–4 (output gradients, propagation through $W^\top$ and f', hidden gradients) are unchanged, which is why the algorithm is so general.

Q133 **A** · lesson 7.4

$(N \times k)(k \times h) = N \times h$, matching H, so it can be multiplied element-wise by $H \odot (1-H)$. If a shape doesn't match, a transpose is missing.

Q134 **C** · lesson 7.4

Every example uses the same bias and $\partial S/\partial b = 1$, so each output unit's bias gradient is its column of $\delta^{(2)}$ summed over the N rows: `d2.sum(0)`. The $1/N$ is already inside $\delta^{(2)}$.

Q135 **C** · lesson 7.5

z_1 influences the cost through y_1 and y_2, so the derivative adds both paths: $[(y_1 - y_1^{(i)}) f'(\bar{y}_1) w_{3,11} + (y_2 - y_2^{(i)}) f'(\bar{y}_2) w_{3,12}]\, f'(\bar{z}_1)\, v_1$. Following one path is the classic mistake.

Q136 **B** · lesson 7.5

Entry j of $W^\top \delta$ is $\sum_k w_{kj} \delta_k$: the sum over every unit that unit j feeds, weighted by the connection. D_k is diagonal, so it only scales; the learning rate isn't part of the gradient.

Q137 **A** · lesson 7.5

Each layer's error signal is built from the one above it, so the brackets computed for layer 2 feed straight into layer 1. That reuse is the whole efficiency of backpropagation; it's exact, not approximate.

Q138 **C** · lesson 7.5

When a node fans out, its gradient is the sum over all downstream paths; dropping some gives a consistently wrong value that a gradient check exposes. $\epsilon \approx 10^{-6}$ works well, and a correct derivation does agree with the check.

Q139 **C** · lesson 7.5

The product contains a D (activation derivatives) for every layer between k and L; with sigmoids each is at most 0.25, so early-layer gradients shrink geometrically. The weights $W^\top$ appear too, so the gradient does depend on them.

Q140 **C** · lesson 7.6

An epoch is one pass through the data: $10{,}000/100 = 100$ mini-batches, one update each. 10,000 would be online (one example per update), 1 would be full-batch.

Q141 **B** · lesson 7.6

Mini-batches give many updates per epoch, each a reasonable gradient estimate, and batched maths runs efficiently on hardware. The estimate is noisy, a learning rate is still needed, and no method guarantees the global minimum.

Q142 **D** · lesson 7.6

Expanding the squares, $\frac{(3+\epsilon)^2-(3-\epsilon)^2}{2\epsilon} = \frac{12\epsilon}{2\epsilon} = 6$, which matches the true derivative $E'(3) = 2 \times 3$. The tempting 9 is the value $E(3)$ itself, not its slope, and 12 forgets to divide by the 2 in 2ϵ.

Q143 **B** · lesson 7.6

Each parameter needs its own pair of perturbed loss evaluations, so a million weights would need two million forward passes per step. Backprop gets all gradients for about one extra pass. The check is accurate and works for any network.

Q144 **B** · lesson 7.6

In one lab, a sign error in a single bias still let the network learn XOR in about half the random starts. Only a gradient check (central differences, relative error below about 10^{-6}) verifies the gradients.

Q145 **B** · lesson 8.1

The most flexible model fits the training points most closely, noise included, so degree 15 has the lowest training error, and yet the worst predictions between the points. Degree 4 generalises best; degree 1 underfits.

Q146 **D** · lesson 8.1

Generalisation is about unseen inputs drawn from the same distribution as the training data. Low training error is only a proxy; working on any distribution is far more than generalisation promises.

Q147 **C** · lesson 8.1

Random labels contain nothing to generalise, so fitting them is pure memorisation. That's why low training error alone proves nothing: you must test on held-out data.

Q148 **B** · lesson 8.1

Generalisation can only be measured on held-out data. The training loss can be near zero for a memorising model, parameter counts don't tell you, and more epochs only show the training loss.

Q149 **D** · lesson 8.1

With 16 parameters and 20 noisy points, the fit bends to pass near each point, noise included, and is free to swing between them. It has too many parameters, not too few.

Q150 **A** · lesson 8.2

Validation data drive every decision other than the weights themselves, and detect overfitting. Gradients come from the training set; the final unbiased estimate comes from the test set, used once.

Q151 **D** · lesson 8.2

Picking the best of 30 test scores turns the test set into a validation set, and the winning score is biased upwards. Compare configurations on validation data and look at the test set once, at the end.

Q152 **D** · lesson 8.2

Scans of the same patient are strongly related; splitting them across sets lets the model recognise the patient rather than the condition, inflating the test score. Group the split by patient.

Q153 **B** · lesson 8.2

In use, the model always predicts the future from the past, so the test must mirror that. A random split lets the model learn from months after the ones it's tested on, which is leakage.

Q154 **A** · lesson 8.2

Each of the 5 folds takes one turn as the validation fold, so every example is validated exactly once and used for training in the other 4 rounds.

Q155 **B** · lesson 8.3

Both errors are high and close together: the model lacks capacity or hasn't trained enough. Overfitting shows a large gap; a small gap alone doesn't mean a good fit if both errors are poor.

Q156 **D** · lesson 8.3

Low training error with much higher validation error is the overfitting signature; the 14-point gap measures it. An underfitting model would have high training error too.

Q157 **B** · lesson 8.3

Early on, both costs fall as the network learns the real pattern; later it fits noise, so training cost keeps falling but validation cost rises. The minimum is where to stop.

Q158 **B** · lesson 8.3

σ^2 is randomness in the targets themselves; no model can predict it. Bias falls with more capacity, variance falls with more data or regularisation, and training error isn't a term of the decomposition.

Q159 **C** · lesson 8.3

Variance measures how much the fitted model changes with the training sample: high-capacity models have high variance. High bias would show as consistently missing the pattern; noise is fixed by the data.

Q160 **D** · lesson 8.4

$(45 - 20)/(70 - 20) = 25/50 = 0.5$. 0.64 divides by 70 instead of the range, 0.9 divides by 50 without subtracting the minimum, and 25 forgets to divide.

Q161 **A** · lesson 8.4

The test rows influence the statistics used to transform the training data, a subtle form of leakage. Compute them on the training set only and apply the same transformation to validation and test data.

Q162 **A** · lesson 8.4

Large inputs give large $|z|$, where $\sigma' \approx 0$, so learning stalls; the loss surface is also badly conditioned. Sigmoids accept any input, and scaling doesn't change the parameter count.

Q163 **A** · lesson 8.4

One-hot encoding gives each colour its own input without implying an order. Coding 1, 2, 3 tells the network blue is "three times" red and green sits between them, which is meaningless.

Q164 **B** · lesson 8.4

Predicting "legitimate" for everything already scores 99%. The confusion matrix, precision and recall show whether it catches any fraud. Consider re-weighting the loss or over-sampling fraud cases.

Q165 **A** · lesson 8.5

Oscillating or jumping cost is the signature of steps that overshoot: reduce η. Raising it makes things worse, and architecture or validation changes don't address unstable steps.

Q166 **D** · lesson 8.5

Memorising 100 examples is easy for a reasonably sized network; failing suggests wrong labels, a broken gradient or a mistaken setting. Overfitting would mean succeeding too well, and more data won't fix a bug.

Q167 **B** · lesson 8.5

Moving from 0.001 to 0.01 matters as much as 0.1 to 1, so a log scale covers the useful range evenly. There's no rule that rates be powers of ten, and smaller rates are usually slower.

Q168 **D** · lesson 8.5

Both errors high means underfitting, so add capacity, train longer, or raise η (and check normalisation). Dropout, weight decay and early stopping fight overfitting; repeated test-set use is leakage.

Q169 **D** · lesson 8.5

A grid's cost grows exponentially and repeats the same few values of each hyperparameter; random sampling tries many different values for the same budget. It still needs validation data and guarantees nothing.

Q170 **D** · lesson 9.1

$W \leftarrow (1-\eta\lambda)W - \eta\,\partial J/\partial W$, and $1 - 0.1 \times 0.01 = 0.999$. 0.99 uses λ alone, 0.9 uses η alone, and 0.001 is $\eta\lambda$ itself, the amount removed rather than kept.

Q171 **C** · lesson 9.1

L1's gradient $\lambda\,\mathrm{sign}(w)$ pulls with constant force even on tiny weights, driving them to exactly zero (sparsity). L2's pull λw fades as w shrinks, so weights get small but rarely reach zero.

Q172 **B** · lesson 9.1

The gradient of $\frac{\lambda}{2}w^2$ is λw: as $w \to 0$, so does the pull, and the weight approaches zero without reaching it. L1's pull stays at size λ, which is why L1 produces exact zeros.

Q173 **A** · lesson 9.1

L1's sparsity switches off unhelpful inputs, a form of feature selection. L2 shrinks all weights but keeps every feature in play; penalising biases doesn't address irrelevant inputs.

Q174 **D** · lesson 9.1

An enormous penalty outweighs the data term, so the weights shrink to zero and the model underfits. $\lambda = 0$ is the unpenalised problem. Biases aren't penalised, so they're free to fit the mean.

Q175 **D** · lesson 9.2

You save the weights whenever the validation cost reaches a new low and restore them at the end. The last epoch has trained past the best point; the lowest training cost usually means the most overfitted model.

Q176 **A** · lesson 9.2

Training continues until 15 epochs pass without improvement: $31 + 15 = 46$. You then restore the weights from epoch 31. Stopping at 31 would require knowing in advance that no later epoch improves.

Q177 **B** · lesson 9.2

Weights start small and grow during training; stopping early keeps them closer to their start, which limits how complex the function becomes. For a linear model it's equivalent to L2. Dropping units is dropout; a loss penalty is L1/L2.

Q178 **A** · lesson 9.2

Validation cost is noisy, especially with mini-batches; patience waits a few epochs before concluding training has stopped helping. Since the best weights are restored, waiting costs only time.

Q179 **D** · lesson 9.2

Stopping at the best validation epoch effectively chooses how long to train, one hyperparameter fewer to tune. It doesn't set the learning rate, width or batch size.

Q180 **C** · lesson 9.3

Kept units are scaled by $1/(1-p) = 2$: $(2 \cdot 2, 0, 2 \cdot 1, 0) = (4, 0, 2, 0)$. $(2, 0, 1, 0)$ forgets the scaling, $(1, 0, 0.5, 0)$ scales the wrong way, and $(4, 8, 2, 6)$ ignores the mask.

Q181 **B** · lesson 9.3

The $1/(1-p)$ scaling during training already keeps the expected activations right, so the full network is used unchanged at test time. Scaling by $1-p$ at test time is the older convention.

Q182 **C** · lesson 9.3

If any unit may vanish, none can depend on particular partners, so each must be useful alone, and units can't jointly fit noise. Dropout also trains an implicit ensemble of sub-networks. It isn't a fix for gradient pathologies.

Q183 **D** · lesson 9.3

In training mode, dropout keeps dropping units, so evaluation is random and worse. `model.eval()` switches dropout off. `zero_grad` affects training, and dropout never belongs on the output layer.

Q184 **B** · lesson 9.3

The older form uses p = keep probability and scales at test time; the lectured inverted dropout uses p = drop probability and scales by $1/(1-p)$ in training. Both preserve expected activations; use the lecture's convention and say which you use.

Q185 **D** · lesson 9.4

Rotating a 6 by 180° turns it into a 9, so the label is wrong. Small shifts, brightness and zoom changes keep the digit's identity, which is the requirement for any augmentation.

Q186 **B** · lesson 9.4

Applying fresh random transformations each mini-batch means the network rarely sees the same example twice. Evaluation must use the real inputs, so validation and test data are never augmented.

Q187 **D** · lesson 9.4

By seeing shifted, cropped or brightened copies with the same label, the network learns that those changes don't matter. Augmentation keeps the labels; it discourages memorisation rather than encouraging it.

Q188 **B** · lesson 9.4

Shifting a recording slightly or adding noise doesn't change which word was spoken. Reversing the audio (or flipping a spectrogram in time) produces a different sound, and swapping labels corrupts the data.

Q189 **C** · lesson 9.4

Regularisation trades a little training performance for better generalisation: the task got harder, so training error rose, and the model generalises better, so validation error fell. That's success, not a bug.

Q190 **D** · lesson 10.1

$32 \times 32 \times 3 = 3{,}072$ inputs, so $3{,}072 \times 100 + 100 = 307{,}300$. 307,200 forgets the biases, 3,172 adds instead of multiplying, and 102,400 ignores the 3 colour channels.

Q191 **D** · lesson 10.1

Each filter has $3 \times 3 \times 3 = 27$ weights plus a bias: $16 \times 27 + 16 = 448$. 432 forgets the biases, 160 ignores the channels, and 49,152 counts one filter copy per pixel, which weight sharing avoids.

Q192 **B** · lesson 10.1

One filter slides across the whole input with the same weights, so a pattern is detected wherever it appears and the parameter count doesn't depend on the image size. Different filters have different weights; full connectivity is the FC layer.

Q193 **C** · lesson 10.1

Thanks to weight sharing, a 3×3×3 filter has 27 weights whether it slides over a 32×32 or a 1000×1000 image; only the feature map gets bigger. The image isn't resized and every pixel is covered.

Q194 **A** · lesson 10.1

Nearby pixels are related (local connectivity) and the same pattern can appear anywhere (weight sharing). Colour and brightness are inputs the filters process, not the structural ideas behind the layer.

Q195 **D** · lesson 10.2

$\lfloor (n + 2p - k)/s \rfloor + 1 = (4 - 3)/1 + 1 = 2$. 4×4 would need padding 1, and 1×1 would need a 4×4 filter. The filter fits in 2 positions along each axis.

Q196 **D** · lesson 10.2

$3 \times 1 + 4 \times 0 + 3 \times (-1) = 0$: no left-to-right change in brightness, so no vertical edge. 10 just adds the sums, 6 forgets the minus sign on the right column, and −6 flips it.

Q197 **C** · lesson 10.2

$(32 + 2 \times 2 - 5)/1 + 1 = 32$: padding of 2 keeps the size unchanged for a 5×5 filter. 28 ignores the padding, and 16 would need stride 2.

Q198 **D** · lesson 10.2

It subtracts the right column from the left, so it's large when the left side is brighter: a vertical edge. A uniform patch gives 0, and a horizontal edge needs row weights such as $(+1, 0, -1)$ down the rows.

Q199 **A** · lesson 10.2

Each filter spans all 3 input channels and sums over them, producing one feature map. 16 filters give 16 maps, which become 16 input channels for the next layer.

Q200 **C** · lesson 10.3

Max pooling keeps the largest value, 4. 2.5 would be average pooling, 10 the sum, and 1 the minimum.

Q201 **C** · lesson 10.3

$(0 + 1 + 2 + 6)/4 = 9/4 = 2.25$. 6 is the max, 9 the sum, and 1.5 averages only the middle two values.

Q202 **D** · lesson 10.3

Non-overlapping 2×2 windows halve each dimension: 16/2 = 8. 15×15 would be stride 1, and 4×4 would be pooling twice.

Q203 **D** · lesson 10.3

Pooling applies a fixed rule (max or average) with no weights or biases to learn. It just summarises each window.

Q204 **C** · lesson 10.3

If the strong response moves but stays inside the same window, the max is unchanged, which gives a degree of translation invariance. Pooling has no weights, and the convolution's output does move with the feature.

Q205 **D** · lesson 10.4

After convolution and pooling extract features, dense layers (with a softmax for classification) turn them into the final output. Edges are detected by early conv layers, pooling shrinks maps, and activations add non-linearity.

Q206 **B** · lesson 10.4

Early filters respond to simple local patterns; middle layers combine them into textures and parts; deep layers respond to whole objects. Class labels come from the final dense layers.

Q207 **B** · lesson 10.4

Convolution is linear, so without a non-linearity a stack of conv layers is equivalent to one. ReLU lets depth build richer features. Shrinking is pooling's job; weight sharing is built into convolution.

Q208 **C** · lesson 10.4

Objects are made of parts, and parts of edges; learning features that mirror that structure means the same detectors serve many images and classes. That's why hand-crafted features are no longer needed, not because labels or memorisation are.

Q209 **B** · lesson 10.4

Each layer-1 unit covers 3×3 pixels, and 3×3 neighbouring layer-1 units together cover $3 + 2 = 5$ pixels in each direction. This receptive field grows with depth (and with pooling). 9×9 would multiply instead of adding.

Q210 **D** · lesson 11.1

Depth lets later layers build on earlier features, which can represent structured functions with far fewer units. It doesn't always reduce parameters, it makes optimisation harder (not easier), and it guarantees nothing about generalisation.

Q211 **C** · lesson 11.1

The connecting weights number $100 \times 100 = 10{,}000$, becoming $200 \times 200 = 40{,}000$: four times as many. Width grows parameters with the product of adjacent widths, which is why wide networks get large fast.

Q212 **B** · lesson 11.1

Signals must pass back through many layers, where gradients can vanish or explode, so training is harder; ReLU, normalisation and residual connections help. Depth adds expressive power and is used well beyond images.

Q213 **D** · lesson 11.1

Tabular data rarely have deep compositional structure, and small datasets overfit easily, so 2–3 dense layers are a good start, tuned on validation data. Very deep or extremely wide networks mostly add trouble.

Q214 **B** · lesson 11.1

Depth can be exponentially more efficient yet harder to optimise; width trains easily but adds parameters and overfitting risk. The universal approximation theorem allows one wide layer but doesn't make it the better design.

Q215 **C** · lesson 11.2

Price is a continuous number: a linear output and a regression loss (MSE, or MAE if outliers matter). Sigmoid and softmax are for classification, binning prices throws away information, and accuracy has no gradient.

Q216 **B** · lesson 11.2

With many categories, one-hot encoding gives 2,000 sparse inputs; an embedding learns a short vector per postcode, so similar areas end up close. Integer codes impose a false order, and location is too informative to drop.

Q217 **A** · lesson 11.2

Prices are right-skewed; on a log scale errors become relative, and a few mansions don't dominate an MSE loss. Networks can output large numbers, the log doesn't make a network's loss convex, and price is numeric.

Q218 **C** · lesson 11.2

Small datasets overfit easily: penalise large weights, stop at the best validation epoch, and use cross-validation to make reliable choices. More depth and epochs increase overfitting, and using the test set for tuning is leakage.

Q219 **B** · lesson 11.2

A model used to price houses before sale can't use information that exists only afterwards; including it would give great test scores and useless predictions. Scaling or encoding doesn't fix leakage.

Q220 **A** · lesson 11.3

Clear old gradients, compute the predictions and loss, backpropagate to fill the gradients, then update the weights. Stepping before backward would use stale or empty gradients.

Q221 **C** · lesson 11.3

CrossEntropyLoss applies log-softmax itself; an extra softmax distorts the gradients. Return logits and apply softmax only for reporting probabilities. MSE is the wrong loss for classification.

Q222 **A** · lesson 11.3

Gradients accumulate across `backward()` calls, so without clearing them each step uses the sum of all previous gradients. It doesn't touch the weights, the mode (that's `train()`/`eval()`), or the learning rate.

Q223 **B** · lesson 11.3

$100 \times 50 + 50 = 5{,}050$ plus $50 \times 10 + 10 = 510$, total 5,560. 5,500 leaves out the biases, 160 counts units instead of weights, and 5,000 counts only the first layer's weights.

Q224 **B** · lesson 11.3

`torch.optim.SGD(model.parameters(), lr=0.01, weight_decay=1e-4)` adds λW to each gradient, which is L2 weight decay. Dropout is a different regulariser, and `eval()` switches dropout off for evaluation.