AI Research Labs

Deep Learning Foundations · Practice test 7 · 30 questions

Backpropagation

How to use this test. Answer all 30 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.

7.1 The Chain Rule and Computational Graphs

Q1 `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

Q2 `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

Q3 `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

Q4 `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

Q5 `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

Q6 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 $\hat{y}_i(1-\hat{y}_i)\,x^{(i)}$

Q7 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 $\delta^\top \delta = \sum_i \delta_i^2$

Q8 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 $(-0.281, -0.562)$

Q9 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

Q10 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

Q11 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 $(y - t)\,y(1-y)$

Q12 `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 $\delta^{(2)}\, w_j^{(2)}\, x_i$

Q13 `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 $+0.0529$

Q14 `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

Q15 `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

Q16 `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 $d \times h$

Q17 `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

Q18 `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 $\delta^{(2)}$

Q19 `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 $k \times h$

Q20 `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

Q21 `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

Q22 `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

Q23 `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

Q24 `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

Q25 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

Q26 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

Q27 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

Q28 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

Q29 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

Q30 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

Answer key

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

Q1 **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.

Q2 **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.

Q3 **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()`.

Q4 **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.

Q5 **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.

Q6 **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.

Q7 **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.

Q8 **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 $\hat{y}(1 - \hat{y})$.

Q9 **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.

Q10 **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.

Q11 **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.

Q12 **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.

Q13 **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 $w_1^{(2)} > 0$.

Q14 **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.

Q15 **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.

Q16 **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.

Q17 **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.

Q18 **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.

Q19 **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.

Q20 **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 $\delta^{(2)}$.

Q21 **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.

Q22 **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.

Q23 **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.

Q24 **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.

Q25 **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.

Q26 **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.

Q27 **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.

Q28 **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.

Q29 **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.

Q30 **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.