

CSED 502
Deep Learning
Winter 2026 Quiz 1

SOLUTIONS

February 18, 2026

Full Name (as on Gradescope): _____

UW Net ID: _____

Question	Score
True/False (28 pts)	
Multiple Choice (72 pts)	
Total (100 pts)	

Welcome to the CSED 502 Quiz!

- The exam is 45 min and is **double-sided**.
- You may use a non-graphing calculator and no other electronic devices.
- One handwritten double sided cheat sheet is allowed.

I understand and agree to uphold the University of Washington Student Conduct Code during this exam.

1 True/False (28 points) - Recommended 15 Minutes

Fill in the circle next to your choice (like this: ●). Explanations are not required.

Each question is worth 4 points. You will receive partial credit if reasonable short explanation is provided, even if the answer is wrong.

1.1 A linear classifier with a 2-dimensional input can classify into at most 4 classes.

- True
 False

SOLUTION:

False - input size doesn't matter, the linear layer can project it to any size

1.2 When using multiclass SVM loss (hinge loss) on a single example, a model that predicts the correct class can have higher loss than a model that predicts an incorrect class.

- True
 False

SOLUTION:

True

Let true class is B (index 1), margin $\Delta = 1$. $L = \sum_{j \neq y} \max(0, s_j - s_y + \Delta)$

- Model 1 (correct): scores $[1, 1.2, 1]$ → predicts B (highest score)
 - Loss = $\max(0, 1 - 1.2 + 1) + \max(0, 1 - 1.2 + 1) = 0.8 + 0.8 = 1.6$
- Model 2 (incorrect): scores $[2, 1.5, 0]$ → predicts A (wrong)
 - Loss = $\max(0, 2 - 1.5 + 1) + \max(0, 0 - 1.5 + 1) = 1.5 + 0 = 1.5$

Model 1 predicts correctly but has higher loss ($1.6 > 1.5$).

1.3 Using softmax cross-entropy loss on a single example, a model that predicts the correct class can have higher loss than a model that predicts an incorrect class.

- True
 False

SOLUTION:

True

Let true class is B (index 1), margin $\Delta = 1$. $L = \sum_{j \neq y} \max(0, s_j - s_y + \Delta)$

- Model 1 (correct): probabilities $[0.31, 0.38, 0.31]$ → predicts B (highest probability)
 - Loss = $-\log(0.38) \approx 0.97$
- Model 2 (incorrect): probabilities $[0.5, 0.4, 0.1]$ → predicts A (wrong)
 - Loss = $-\log(0.4) \approx 0.92$

Model 1 predicts correctly but has higher loss ($0.97 > 0.92$).

- 1.4 For a single linear layer with softmax cross-entropy loss, two SGD steps (no momentum) on the same single training example produce the same final weights as one step with double the learning rate.
- True
 - False

SOLUTION:

False. The gradient changes after the first update because the softmax probabilities depend on the current weights

- 1.5 If a node receives gradients from multiple upstream paths during backpropagation, those gradients are multiplied.
- True
 - False

SOLUTION:

False, they are added

- 1.6 While training on CIFAR (a dataset with 10 equally represented classes in both test and train sets), you accidentally set the ground truth label of all of your train images to be Class 1. After training on this train set, you would expect your test accuracy to be essentially 0%
- True
 - False

SOLUTION:

False - you would expect 10%

- 1.7 Pooling layers reduce the spatial dimensions of feature maps (aka activation maps)
- True
 - False

SOLUTION:

True, see slides

2 Multiple Choice (40 points) - Recommended 30 Minutes

Fill in the circle next to the letter(s) of your choice (like this: ●). Explanations are not required. Choose ALL options that apply.

Question 2.1 - 2.4 are worth 14 points, 2.5 is worth 16 points and the answer may contain multiple correct options, or none at all. Selecting all of the correct options and none of the incorrect options will get full credit. For questions with multiple correct options, each incorrect or missing selection gets a 4-point deduction (up to the full points for each question).

2.1 Which of the following produce **zero-centered outputs** given mean-zero normal inputs? Select all that apply.

- A: Sigmoid
- B: Tanh
- C: ReLU
- D: Batch Normalization (with $\beta = 0$)
- E: Layer Normalization (with $\beta = 0$)
- F: Softmax

SOLUTION:

B, D, E

2.2 Consider a neural network whose parameters require **M bytes** of storage. Assume that:

- Each additional value stored (e.g., gradients, optimizer state) requires the same memory as a parameter
- Gradients for all parameters are computed and stored during backpropagation before the optimizer step (as in PyTorch/TensorFlow autograd)

Select all statements that are TRUE about the **peak memory** required to store parameters, gradients, and optimizer state. (*Ignore memory for activations.*)

- A: Inference requires approximately M
- B: Inference requires approximately 2M
- C: Training with SGD (no momentum) requires approximately M
- D: Training with SGD (no momentum) requires approximately 2M
- E: Training with SGD (no momentum) requires approximately 3M
- F: Training with Adam requires approximately 4M
- G: Training with Adam requires approximately 5M

SOLUTION:

A, D, F. **Inference:** Only parameters are needed. Peak: M .

Training with SGD: Autograd stores gradients for all parameters during backprop before the optimizer step. Peak: $2M$ (parameters + gradients).

Training with Adam: Same as SGD, plus first moment and second moment estimates for all parameters. Peak: $4M$ (parameters + gradients + first moment + second moment).

2.3 Which of the following could have **different behavior** during training vs. inference? Select all that apply.

- A: Dropout
- B: Batch Normalization
- C: Layer Normalization
- D: ReLU
- E: Softmax
- F: Max Pooling

SOLUTION:

A, B. Dropout randomly zeroes activations during training but is disabled (or scaled) during inference. Batch Normalization uses batch statistics (mean/variance computed from the current batch) during training, but uses running averages computed during training at inference time. Layer Normalization (C) normalizes across features within a single example, so it behaves the same at train and test. ReLU (D), Softmax (E), and Max Pooling (F) all behave identically during training and inference.

2.4 Given a **7x7 input image**, which of the following convolution configurations will **preserve the spatial dimensions** (i.e., produce a 7x7 output)? Select all that apply.

- A: 3x3 filter, stride 1, padding 1
- B: 5x5 filter, stride 1, padding 2
- C: 3x3 filter, stride 1, padding 0
- D: 5x5 filter, stride 1, padding 1
- E: 7x7 filter, stride 1, padding 3
- F: 3x3 filter, stride 2, padding 1

SOLUTION:

A, B, E. Using the formula $\frac{N+2P-F}{S} + 1$:

- (A) $\frac{7+2(1)-3}{1} + 1 = \frac{6}{1} + 1 = 7 \checkmark$
- (B) $\frac{7+2(2)-5}{1} + 1 = \frac{6}{1} + 1 = 7 \checkmark$
- (C) $\frac{7+2(0)-3}{1} + 1 = \frac{4}{1} + 1 = 5$
- (D) $\frac{7+2(1)-5}{1} + 1 = \frac{4}{1} + 1 = 5$
- (E) $\frac{7+2(3)-7}{1} + 1 = \frac{6}{1} + 1 = 7 \checkmark$
- (F) $\frac{7+2(1)-3}{2} + 1 = \frac{6}{2} + 1 = 4$

2.5 You have a dataset of RGB images, each of size $(H, W, 3)$. For storage efficiency, you flatten each image into a 1D vector of length $H*W*3$. Note that if you reshape one of these vectors back to $(H, W, 3)$, you recover the original image exactly.

A malicious actor has sabotaged your pipeline. Before anyone noticed, they applied a fixed random permutation to every flattened vector in both the training and test sets. The same permutation was used for every image, and the original ordering has been lost. If you reshape one of the permuted vectors back to $(H, W, 3)$, you get a scrambled image, not the original.

You now have two versions of the dataset:

- **Original dataset:** flattened vectors of length $H*W*3$ with the natural pixel ordering.
- **Permuted dataset:** flattened vectors of length $H*W*3$ with the shuffled ordering.

You want to understand which architectures are hurt by this sabotage. Consider the four models below. Each first reshapes its input vector, then applies an operation, before passing the result to further layers and a loss function. All operators and functions in the table including $@$, $*$, $.reshape()$, and $.sum()$ carry their standard NumPy meanings.

Model	Reshape	Operation
A	<code>x = input.reshape(H, W, 3)</code>	output of 3×3 convolution on x
B	<code>x = input.reshape(H, W, 3)</code>	output of 1×1 convolution on x
C	<code>x = input.reshape(1, H*W*3)</code>	output = $x @ W_C$ with $W_C.shape = (H*W*3, Z)$
D	<code>x = input.reshape(H*W, 3, 1)</code>	$X = x * W_D$ output = $X.sum(axis=0).sum(axis=0)$ with $W_D.shape = (H*W, 3, Z)$

For each model, suppose you train and test it in two separate settings:

- **Original setting:** train on the original training set, test on the original test set.
- **Permuted setting:** train on the permuted training set, test on the permuted test set.

Which model(s) would you generally expect to have **worse test performance** in the permuted setting compared to the original setting?

- A: Model A
- B: Model B
- C: Model C
- D: Model D

SOLUTION:

AB. Models A and B are both affected because they rely on spatial structure. The 3×3 convolution (Model A) assumes that neighboring pixels are spatially related, and the 1×1 convolution (Model B)

assumes that the 3 channels at each spatial location correspond to meaningful RGB values. After the permutation, both assumptions are broken: adjacent entries in the reshaped array are no longer true spatial neighbors, and the three values grouped into each “pixel” are no longer the R, G, B channels of the same location.

Models C and D are unaffected. Both are equivalent to a fully-connected linear layer applied to all $H*W*3$ input values, which makes no assumptions about spatial ordering. A permutation of the inputs can be perfectly compensated for by a corresponding permutation of the weight matrix rows, so these models can learn equally expressive representations on either dataset.