CSE 493 G1 / 599 G1
Deep Learning
Spring 2026 Exam

SOLUTIONS

April 28, 2026

Full Name: _____

UW Net ID: _____

Question		Score
True/False	(18 pts)	
Multiple Choice	(20 pts)	
Backpropagation	(28 pts)	
KNN	(8 pts)	
CNN	(18 pts)	
Tokenization	(8 pts)	
Total	(100 pts)	

Welcome to the CSE 493 G1 / 599 G1 Exam!

- The exam is 80 min and is **double-sided**.
- No electronic devices are allowed.

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

Mean: 64.1
Median: 65.0
Stdev: 14.7

1 True or False (18 points) - Recommended 15 Minutes

Fill in the circle next to your choice (like this: ●). No explanations are required. Each question is worth 3 points. Fill in exactly one of True or False.

1. **(3 points)** Changing the distance metric used by a 1-nearest-neighbor classifier (e.g., from L_1 to L_2) can change which training image is returned as the nearest neighbor of a given test image, and therefore change the predicted class.
   ◯ True
   ◯ False

   Answer: True.

   Explanation: L_1 and L_2 rank neighbors differently because they weight coordinate-wise differences differently: L_1 sums $|x_i - y_i|$, L_2 sums $(x_i - y_i)^2$. A single large coordinate disagreement is penalized *quadratically* by L_2 but only *linearly* by L_1, so the closest neighbor can differ. Concrete example: test point at the origin with training points $A = (3, 0)$ and $B = (2, 2)$. Under L_1: A is at distance 3, B at 4, so A wins. Under L_2: A is at distance 3, B at $\sqrt{8} \approx 2.83$, so B wins. Same data, different metric, different neighbor, potentially different predicted class.

2. **(3 points)** For a linear classifier with scores $s = Wx + b$ and prediction $\hat{y} = \arg\max_i s_i$, replacing W with αW and b with αb for some fixed positive constant $\alpha > 0$ leaves the predicted class $\hat{y}$ unchanged for every input x.
   ◯ True
   ◯ False

   Answer: True.

   Explanation: The new scores are $s' = \alpha(Wx + b) = \alpha s$. Because $\alpha > 0$, the ordering of the entries is preserved: $s'_i > s'_j \Leftrightarrow s_i > s_j$. The arg max is therefore identical to that of s, so $\hat{y}$ does not change for any x. Note that this would fail if $\alpha < 0$ (which would reverse the ordering) or if we scaled only W without b (which would bias the rescaling in an input-dependent way).

3. **(3 points)** During backpropagation through an add gate $z = x + y$, the upstream gradient $\partial L/\partial z$ is passed *unchanged* to both inputs x and y.
   ◯ True
   ◯ False

   Answer: True.

   Explanation: The local partial derivatives of $z = x + y$ are $\partial z/\partial x = 1$ and $\partial z/\partial y = 1$, regardless of the values of x and y. By the chain rule, $\partial L/\partial x = (\partial L/\partial z) \cdot 1$ and similarly for y. So the add gate acts as a gradient "distributor" that copies the upstream gradient identically to each input. This is in contrast to the multiply gate (which swaps and scales), the max gate (which routes all gradient to the max input), and the subtract gate (which flips the sign on one input).

4. **(3 points)** If a network with Batch Normalization is trained using batch size 1, then each BN layer's output is effectively just the learned bias β and carries no information about the input x.
   ◯ True
   ◯ False

Answer: True.

Explanation: With batch size 1, the batch statistics reduce to $\mu_B = x$ (the single example itself) and $\sigma_B^2 = 0$. The normalized value is

$$\hat{x} = \frac{x - x}{\sqrt{0 + \epsilon}} = 0,$$

so the BN output is $y = \gamma \cdot 0 + \beta = \beta$. The forward pass becomes a constant, independent of x. This is one of the practical reasons BN fails with very small batches, and why LayerNorm or GroupNorm are used in those settings.

5. **(3 points)** When using the softmax cross-entropy loss, if you scaling all the scores by a positive constant $\alpha > 0$ (i.e., replacing the score vector s with αs for any $\alpha > 0$), it leaves the loss unchanged.
◯ True
◯ False

Answer: False.

Explanation: Softmax is *shift*-invariant (adding the same constant to every score leaves the probabilities unchanged), but it is *not* scale-invariant. Scaling by α changes the "temperature" of the distribution:

- $\alpha > 1$ sharpens the output toward one-hot,
- $\alpha < 1$ flattens it toward uniform,
- $\alpha \to 0$ collapses to the uniform distribution.

Since the probabilities change, the cross-entropy loss $-\log p_{y^\star}$ changes as well. This is in contrast to a plain linear classifier's $\arg\max$, which *is* invariant under positive scaling: the argmax depends only on the ordering of the scores, while softmax depends on their absolute differences.

6. **(3 points)** L1 regularization ($\lambda\|w\|_1$) tends to drive unimportant weights to exactly zero at the optimum, whereas L2 regularization ($\lambda\|w\|_2^2$) tends to keep them small but nonzero.
◯ True
◯ False

Answer: True.

Explanation: **Geometric view.** The L1 "ball" $\{w : \|w\|_1 \leq r\}$ has corners along the coordinate axes. The hyperplane of equally-good data-fitting solutions typically first touches this ball at a corner, which corresponds to some coordinates being exactly 0. The L2 ball is a smooth sphere with no corners, so the contact point is generically at a point where all coordinates are small but nonzero.

Algebraic view. For a coordinate w_j near zero, the L1 penalty's contribution to the gradient has constant magnitude λ (from the subgradient of $|w_j|$), which can hold w_j exactly at 0 whenever the data gradient at zero is smaller than λ. By contrast, the L2 penalty's contribution, $2\lambda w_j$, shrinks linearly as $w_j \to 0$, so it can push weights toward zero but never pin them there exactly.

This is the classical "sparse vs. spread" contrast between L1 and L2 and is the reason L1 is often used for feature selection.

2 Multiple Choices (20 points) - Recommended 15 Minutes

Fill in the circle next to the letter(s) of your choice (like this: ●). No explanations are required. **Choose ALL options that apply.**

Each question is worth 5 points. For questions with one or more correct options, selecting all of the correct options and none of the incorrect options gets full credit; each incorrect or missing selection gets a 2.5-point deduction (up to 5 points). For multi-part questions (labeled (a), (b)), each part is worth 2.5 points.

1. **(5 points)** You want to pick the best k for a k-NN classifier. You have 1000 labeled training images and 500 labeled test images. Consider the following protocols:

   - **P1:** Try $k \in \{1, 3, 5, \ldots\}$ on the training set; pick the k that *minimizes training error*. Report test accuracy on the 500 test images.
   - **P2:** Run 5-fold cross-validation on the training set; pick the k that minimizes average validation error. Report test accuracy on the 500 test images.
   - **P3:** Run 5-fold cross-validation on the *combined* train+test set (1500 images); pick the best k and report accuracy on the same combined set.

   Which of the following statements is TRUE? (Choose all that apply.)

   ◯ A: P1 is a principled protocol because it uses only the training set for hyperparameter selection.

   ◯ B: P2 is preferable to P1 because P1 will essentially always select $k = 1$. **CORRECT**

   ◯ C: P2 is inferior to P1 because 5-fold cross-validation costs roughly 5× more compute without changing which k is ultimately selected.

   ◯ D: The test-set accuracy reported under P2 is a reasonable estimate of how the classifier will perform on unseen data. **CORRECT**

   ◯ E: None of the above.

   Answer: B and D.

   Explanation:

   - **(A)** FALSE. Training error of k-NN with $k = 1$ is always 0 (each point is its own nearest neighbor), so P1 always selects $k = 1$. That is not a principled protocol for choosing k.
   - **(B)** TRUE (matches the reasoning above: P2 holds out validation folds that are not used by the classifier at fit time, so different k values produce genuinely different validation errors).
   - **(C)** FALSE. P1 selects k by minimizing training error, which (as explained in (A)) always picks $k = 1$. P2 picks k by minimizing validation error and generally selects a *different* (and better-generalizing) k. The extra compute in P2 buys you a fundamentally different, more principled answer, not the same one at higher cost.
   - **(D)** TRUE. P2's test set is held out from *all* hyperparameter selection, so the final 500-image test number is a clean estimate of generalization.
   - **(E)** FALSE since (B) and (D) are true.

2. **(5 points)** Let $X \in \mathbb{R}^{2\times 3}$. Consider two *separate* layers:

   - **Op 1 (linear):** $Y_1 = XW$ where $W \in \mathbb{R}^{3\times 4}$, so $Y_1 \in \mathbb{R}^{2\times 4}$.
   - **Op 2 (elementwise):** $Y_2 = \mathrm{ReLU}(X)$, so $Y_2 \in \mathbb{R}^{2\times 3}$.

Flatten outputs (e.g. $\mathbb{R}^{2\times4}$) and inputs (e.g. $\mathbb{R}^{2\times3}$) (row-major) before forming Jacobians, so $\partial Y/\partial X$ has shape (#outputs) $\times$ (#inputs). Which statements are TRUE? (Choose all that apply.)

◯ A: For Op 1, $\partial Y_1/\partial X$ is a 8×6 matrix, and at most 24 of its 48 entries can be nonzero (the rest are structural zeros). **CORRECT**

◯ B: For Op 2, $\partial Y_2/\partial X$ is a 2×3 matrix because the function is element-wise.

◯ C: For Op 2, $\partial Y_2/\partial X$ is a 6×6 matrix, and at most 6 of its entries can be nonzero. **CORRECT**

◯ D: For Op 1, every entry of $\partial Y_1/\partial X$ can be non-zero; there are no structural zeros.

◯ E: None of the above.

Answer: A and C.

Explanation: **Op 1, linear layer.** $Y_1[n,j] = \sum_{k=1}^{3} X[n,k] \cdot W[k,j]$. So

$$\frac{\partial Y_1[n,j]}{\partial X[n',k]} = \begin{cases} W[k,j] & \text{if } n = n', \\ 0 & \text{otherwise.} \end{cases}$$

Flattened shape: Y_1 has $2 \cdot 4 = 8$ entries, X has $2 \cdot 3 = 6$ entries, so the Jacobian is $8 \times 6 = 48$ entries. Nonzero structure: fix row n on both output and input; inside that block there are $4 \cdot 3 = 12$ entries. Across $n = 1, 2$ that's $2 \cdot 12 = 24$ possibly-nonzero entries. The other 24 are structurally zero (outputs in row n do not depend on inputs in row $n' \neq n$).

So (A) is TRUE and (D) is FALSE.

Op 2, element-wise ReLU. $Y_2[n,k] = \max(0, X[n,k])$. Each output entry depends only on the corresponding input entry:

$$\frac{\partial Y_2[n,k]}{\partial X[n',k']} = \begin{cases} \mathbb{1}[X[n,k] > 0] & \text{if } (n,k) = (n',k'), \\ 0 & \text{otherwise.} \end{cases}$$

The full flat Jacobian has shape (#outputs) $\times$ (#inputs) $= 6 \times 6 = 36$ entries. Only the diagonal (6 entries) can be nonzero.

So (C) is TRUE and (B) is FALSE, the *physical storage* we'd use in a framework is a 2×3 mask (one derivative per position), but the formal Jacobian is 6×6.

(E) False since (A) and (C) are correct.

3. **(5 points)** A convolution layer is applied to a single input image of shape $(C, H, W) = (3, 224, 224)$. It is a Conv2D with 10 filters, each with a kernel size 5×5, stride 3, padding 3.

Use the standard $(H + 2P - F)/S + 1$ formula for conv, and assume conv layers have learnable bias. Answer both parts below; each part has exactly one correct option.

What is the output shape (channels $\times$ height $\times$ width) *after applying the convolution layer*?

◯ A: $3 \times 75 \times 75$.

◯ B: $10 \times 74 \times 74$.

◯ C: $10 \times 75 \times 75$.

◯ D: $10 \times 76 \times 76$. **CORRECT**

◯ E: None of the above.

Answer: (a) D.

Explanation: **Shape, layer by layer.** Input: $3 \times 224 \times 224$.

5×5 conv, stride 3, pad 3.

$$H_{\text{out}} = \frac{224 + 2 \cdot 3 - 5}{3} + 1 = \frac{225}{3} + 1 = 75 + 1 = 76.$$

Output: $10 \times 76 \times 76$. ✓

4. **(5 points)** For the convolution filter in the previous question, what is the *total* number of learnable parameters in this convolution layer?

   Again, it is a Conv2D with 10 filters, each with a kernel size 5×5, stride 3, padding 3.

◯ A: 750.

◯ B: 760. **CORRECT**

◯ C: 75.

◯ D: 76.

◯ E: None of the above.

Answer: (b) B.

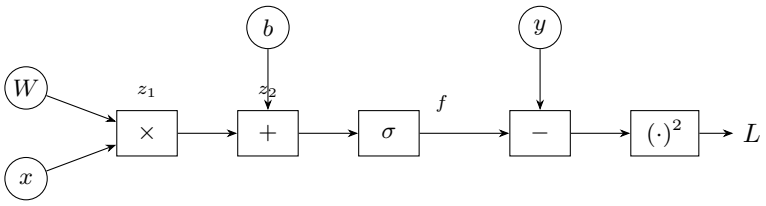
Explanation: Each filter has $C_{\text{in}} \cdot F^2 = 3 \cdot 5 \cdot 5 = 75$ weights, plus 1 bias. With 32 output filters: $10 \cdot (75 + 1) = 10 \cdot 76 = 760$ params. ✓

3 Backpropagation (28 points + 5 extra credit) - Recommended 25 Minutes

Please make sure to write your answer only in the provided space.

3.1 A scalar sigmoid unit.

Consider the scalar model $f = \sigma(Wx + b)$ with squared-error loss $L = (f - y)^2$ and where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid activation function. Its computation graph is shown below.



(a) (8 points) For $x = 1$, $W = 0$, $b = 0$, $y = 1$, fill in the forward values and the gradient at each node. You may use $\sigma(0) = 1/2$ and $\sigma'(0) = 1/4$.

Forward	**value**	**Backward**	**value**
z_1	0	$\partial L/\partial f$	-1
z_2	0	$\partial L/\partial z_2$	$-1/4$
f	$1/2$	$\partial L/\partial W$	$-1/4$
L	$1/4$	$\partial L/\partial b$	$-1/4$

(b) (2 points) Derive $\partial f/\partial W$ analytically. You may use $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

SOLUTION:

(a) Filled in the table above.

(b) $\dfrac{\partial f}{\partial W} = \sigma'(Wx + b) \cdot x = \sigma(Wx + b)\,(1 - \sigma(Wx + b)) \cdot x.$

3.2 The sigmoid-derivative curve.

Fig. 1 plots the $\partial f/\partial W$ formula you derived in 3.1(b), evaluated at $x = 1$, $b = 0$, as a function of W.

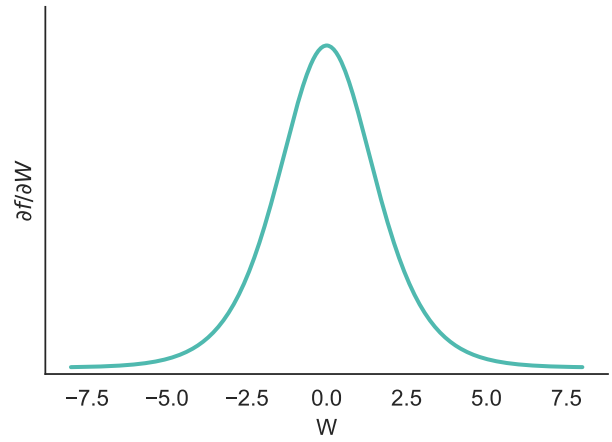


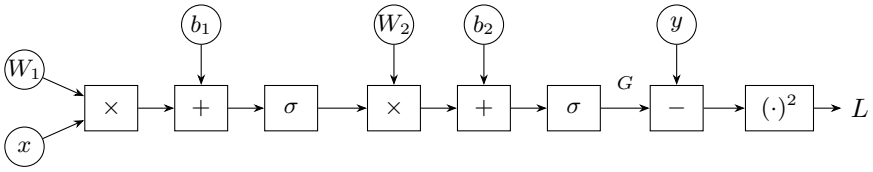
Fig. 1

(a) (3 points) State the gradient descent update rule for W, and explain why large initial values of $|W|$ lead to poor loss minimization. Reference Fig. 1 in your answer.

SOLUTION:
Update: $W \leftarrow W - \eta\, \partial L/\partial W$. Large $|W|$ pushes $z = Wx + b$ deep into the sigmoid tails, where Fig. 1 shows $\sigma'(z) \approx 0$; via the chain rule $\partial L/\partial W = 2(f - y)\, \sigma'(z)\, x$, the update vanishes — the "dead-gradient" regime, in which W barely moves per step.

3.3 Two-layer scalar composition.

Now consider the two-layer composition $G = \sigma(W_2\,\sigma(W_1 x + b_1) + b_2)$ with the same squared-error loss $L = (G - y)^2$. The computation graph is shown below; for some choice of input and weights, the gradients for this network are $\partial L/\partial W_1 \approx -2.3 \times 10^{-5}$ and $\partial L/\partial W_2 \approx -1.2 \times 10^{-4}$ (for $x = 1$, $W_1 = 2$, $b_1 = 1$, $W_2 = 4$, $b_2 = 1$, $y = 1$).



(a) (3 points) Explain why $|\partial L/\partial W_1|$ is much smaller than $|\partial L/\partial W_2|$. Tie your explanation back to 3.2.

Hint: You should not have to compute the gradient to answer this question.

SOLUTION:
Backprop to W_1 traverses one extra sigmoid relative to W_2, picking up an additional σ' factor. By 3.2, σ' saturates to near-zero whenever its argument has large magnitude, so each extra layer multiplies the upstream gradient by a small number $< 1/4$ — the "vanishing gradient" effect. With more layers this factor compounds further.

3.4 Fully-connected variant and weight initialization.

Consider a fully-connected variant of the two-layer model from 3.3:

$$h = \sigma(\mathbf{W}_1 x + b_1), \qquad G = \sigma(\mathbf{W}_2 h + b_2),$$

where the input $x \in \mathbb{R}^2$ is now a 2-dimentional feature vector, $\mathbf{W}_1 \in \mathbb{R}^{4\times 2}$ and $\mathbf{W}_2 \in \mathbb{R}^{1\times 4}$ are weight matrices (boldface to distinguish them from the scalar W_1, W_2 of 3.3), $b_1 \in \mathbb{R}^4$, $b_2 \in \mathbb{R}$, and $h \in \mathbb{R}^4$ is the hidden activation. The scalar chain from 3.3 corresponds to a single *stream* of this network — one entry of $\mathbf{W}_1$ times one entry of $\mathbf{W}_2$, shown as the highlighted path in Fig. 2 — which reduces exactly to the scalar 3.3 model only when every off-path entry of $\mathbf{W}_1$ and $\mathbf{W}_2$ is zero.

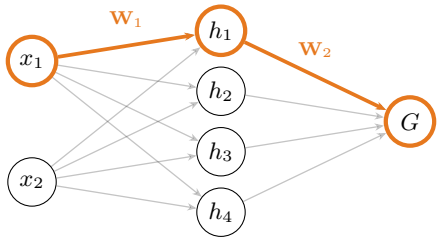


Fig. 2

We train this network on a binary classification task. The four loss curves (A–D) in Fig. 3 correspond to four different weight initializations ($\eta = 0.01$ for 5000 epochs): A — all zeros; B — Gaussian ($W_{ij} \sim \mathcal{N}(0, 1)$); C — large magnitude ($W_{ij} \sim \mathcal{N}(0, 10^2)$); D — symmetric ($W_{ij} = 1$ for every entry of $\mathbf{W}_1$ and $\mathbf{W}_2$).

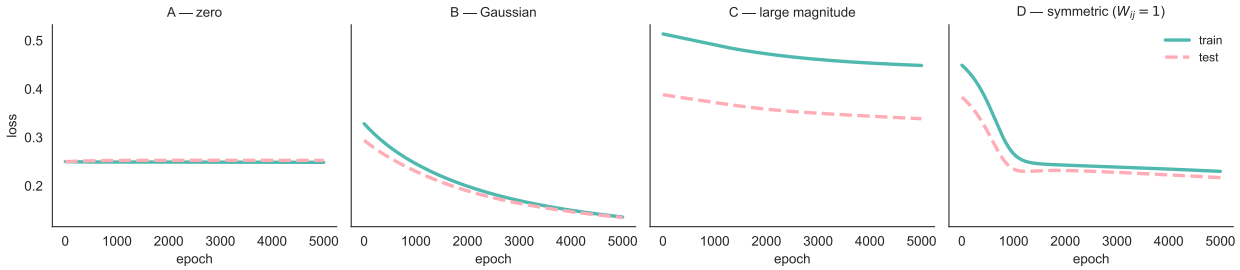


Fig. 3

(a) (4 points) Fill in the matrix shape of each intermediate variable in the forward pass.

$z_1 = \mathbf{W}_1 x + b_1$	shape of z_1: _____
$h = \sigma(z_1)$	shape of h: _____
$z_2 = \mathbf{W}_2 h + b_2$	shape of z_2: _____
$G = \sigma(z_2)$	shape of G: _____

(b) (8 points) Explain each loss curve:

(i) *Curve A:* Why does the loss stay constant?

(ii) *Curve B:* Is this a good loss curve? Why does this one decrease while the others don't?

(iii) *Curve C:* Why does the loss start so high and stay high?

(iv) *Curve D:* Why does the loss decreases initially but plateaus so quickly? What might explain this?

SOLUTION:

(a) $z_1 \in \mathbb{R}^4$, $h \in \mathbb{R}^4$, $z_2 \in \mathbb{R}$, $G \in \mathbb{R}$.

(b)

- *(i) Curve A.* All weights are 0, so the first-layer pre-activation z_1 is 0 regardless of x, and every hidden unit is identical. Worse, the gradient at $\mathbf{W}_1$ carries a factor of $\mathbf{W}_2 = 0$ (3.3a), so $\mathbf{W}_1$ never updates. Only b_2 can move; the loss flat-lines at the constant output $G \approx 0.5$.
- *(ii) Curve B.* Yes — this is the healthy case. With $\mathbf{W}_{ij} \sim \mathcal{N}(0, 1)$, the pre-activations z sit near 0 where $\sigma'(z)$ is at its peak 1/4 (3.2), so gradient flows cleanly through both layers (3.3 stream argument). The other curves all fail: A has zero gradient, C is saturated, and D has all hidden units locked together.
- *(iii) Curve C.* With $\mathbf{W}_{ij} \sim \mathcal{N}(0, 100)$ the pre-activations z are huge, so σ saturates and $\sigma'(z) \approx 0$ (3.2). The 3.3 stream argument then multiplies two near-zero σ' factors together, killing the gradient at both layers; only b_2 drifts. The output is also a nearly-constant near-saturated value, which gives a high MSE.
- *(iv) Curve D.* With $\mathbf{W}_{ij} = 1$ everywhere, every hidden unit sees the same pre-activation $z_j = x_1 + x_2 + b_1$ and receives the same gradient. The four units therefore update in lockstep — the network has the expressive capacity of a single hidden unit, which is enough to descend at first but not enough to fit the data well, so it plateaus above curve B.

3.5 LayerNorm (Extra Credit).

Consider an extension of the model in 3.4 where a *LayerNorm* is inserted after the first linear layer and before the hidden sigmoid:

$$z_1 = \mathbf{W}_1 x + b_1, \qquad \hat{z}_1 = (z_1 - \mu)/s, \qquad h = \sigma(\hat{z}_1),$$

where μ, s are the per-sample mean and standard deviation of z_1 across the four hidden units. The augmented single-stream computation graph is shown below.

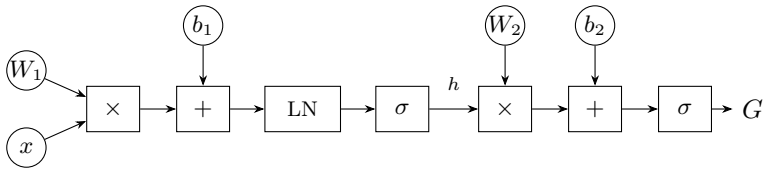


Fig. 4 shows training loss on the moons data under the same poor (large-magnitude) initialization as 3.4(C), with and without LN (left panel). The right panel shows the per-entry gradient magnitude at $\mathbf{W}_1$ and $\mathbf{W}_2$, captured at epoch 1000 (dashed grey line in the left panel), when the no-LN model is only slowly improving.

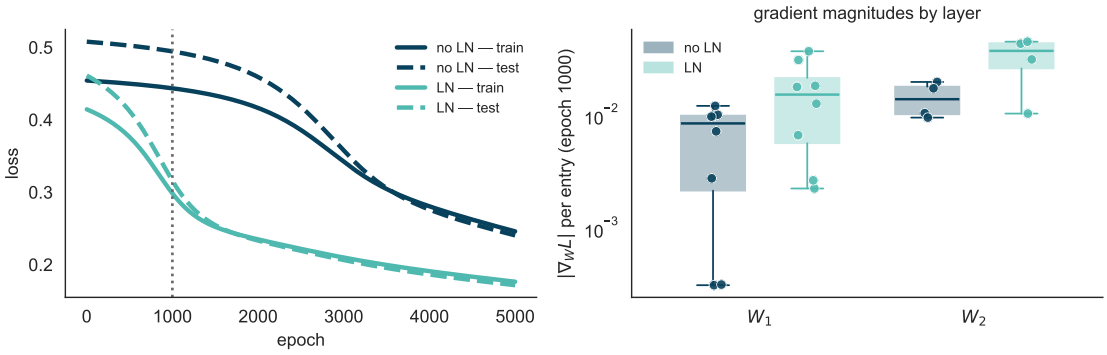


Fig. 4

(a) (1 point extra credit) Assume the gradient flowing backward into the LN node from above is g (i.e. $\partial L/\partial \hat{z}_1 = g$). What is the gradient flowing out of LN toward the linear layer below ($\partial L/\partial z_1$)? Answer in terms of g and s.

(b) (2 points extra credit) Why does the LN layer lead to better loss curves, even when $\mathbf{W}_1$ is initialized to be a really large value? Refer to Fig. 1.

(c) (1 point extra credit) Using the right panel of Fig. 4, why does the LN variant converge faster than the no-LN model?

(d) (1 point extra credit) Using the right panel of Fig. 4, why does LN raise the gradient magnitude at $\mathbf{W}_2$ as well as $\mathbf{W}_1$?

SOLUTION:

(a) $\hat{z}_1 = (z_1 - \mu)/s$ with μ, s treated as constants gives $\partial \hat{z}_1 / \partial z_1 = 1/s$, so the downstream gradient is g/s.

(b) Fig. 1 shows σ' peaked at 1/4 when its argument is 0 and decaying to 0 in the tails. Without LN, $|z_1|$ scales with $|\mathbf{W}_1|$: a really large init lands z_1 in the dead tails (3.4 curve C), $\sigma'(z_1) \approx 0$, and the gradient at $\mathbf{W}_1$ vanishes. With LN, $\hat{z}_1$ is mean-0, unit-variance regardless of the size of $\mathbf{W}_1$, so $\sigma'(\hat{z}_1)$ stays near its peak and gradients flow normally — the loss can decrease.

(c) Right panel: LN gradients sit $\sim 2\times$ above no-LN at both layers. Under $W \leftarrow W - \eta \nabla_W L$, larger gradients give larger per-step updates, so LN accumulates more useful weight changes per epoch and the loss descends faster.

(d) Recall $\partial L / \partial \mathbf{W}_2 = (\partial L / \partial z_2) \cdot h$, where $\partial L / \partial z_2 = (\partial L / \partial G) \cdot \sigma'(z_2)$ and $z_2 = \mathbf{W}_2 h + b_2$.

Without LN, the saturated z_1 forces every h_j to be either ≈ 0 or ≈ 1. Two compounding effects shrink $\partial L / \partial \mathbf{W}_2$:

- the entries of h that pin to 0 contribute zero to the $\mathbf{W}_2$ gradient (since the gradient is multiplied by h);
- summing large $\mathbf{W}_2$ entries against a near-binary h produces a large $|z_2|$, so $\sigma'(z_2) \approx 0$ at the output sigmoid, killing $\partial L / \partial z_2$ as well.

With LN, h is spread around 0.5 rather than pinned at the extremes, so every $\mathbf{W}_2$ entry receives a non-zero contribution and the second sigmoid stays in its responsive region. Saturation in $\mathbf{W}_1$ therefore propagates to the $\mathbf{W}_2$ gradient too, and removing it lifts both.

4 KNN (8 points) - Recommended 5 Minutes

Please make sure to write your answer only in the provided space. A *K-nearest-neighbors* (KNN) classifier assigns to each input the majority label among its K closest points in the training set (by Euclidean distance). Unlike the sigmoid MLP from 3.4, KNN has no trainable parameters — it simply stores the training data and computes distances at prediction time.

Fig. 5 shows the decision boundaries of four classifiers on the moons training data: the MLP from 3.4(B) and three KNN classifiers. Which is which has been anonymized as "Model A"–"Model D". The training set contains $N_0 = 106$ class-0 points and $N_1 = 94$ class-1 points ($N = N_0 + N_1 = 200$).

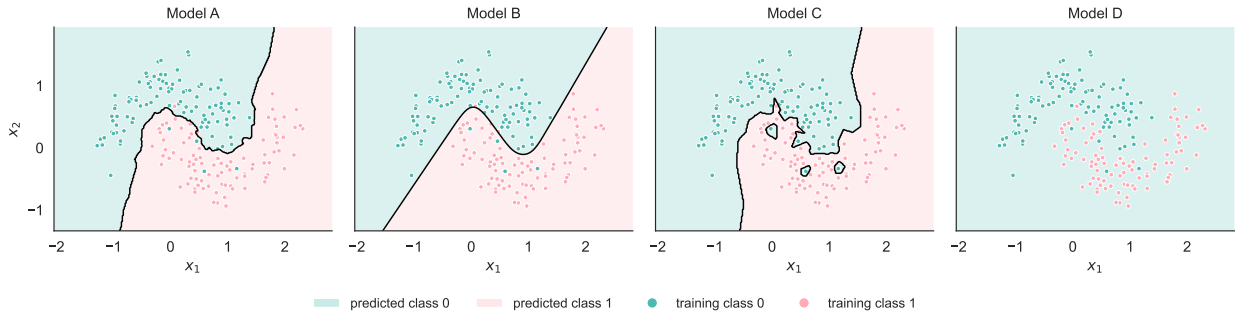


Fig. 5

(a) (6 points) Three of the four panels (A, B, C) show KNN classifiers with $K \in \{1, 5\}$ together with the MLP. Match each of Models A, B, C to one of {MLP, KNN with $K = 1$, KNN with $K = 5$}, and justify your assignments from the shape of the boundaries.

(b) (2 points) Panel D shows a KNN classifier with $K = N = 200$. Explain why its prediction is constant across the plane, and use the class balance stated above to determine which class it predicts.

SOLUTION:

(a) A = KNN $K = 5$; B = MLP; C = KNN $K = 1$.

- *C (K=1):* most jagged, each training point owns a small Voronoi cell; label noise shows up as tiny outlier islands.
- *A (K=5):* piecewise-linear with moderate pockets — local voting smooths the $K = 1$ cells but still can't produce a continuous curve.
- *B (MLP):* single smooth curve — the 0.5 level set of $\sigma(\mathbf{W}_2\sigma(\mathbf{W}_1 x + b_1) + b_2)$ is necessarily continuous and is sculpted by only four hidden sigmoids.

(b) With $K = N$ every query sees the entire training set, so the neighbor average equals the global mean label $N_1/N = 94/200 = 0.47 < 0.5$ at every point. Majority vote therefore returns class 0 everywhere — KNN at $K = N$ reduces to the constant majority-class predictor.

5 CNN (18 points) - Recommended 15 Minutes

Sunny is training a binary image classifier on 32×32 RGB images. Her first architecture applies a single convolutional layer with 32 filters of size 3×3 (stride 1, padding 1), followed by Flatten and a fully-connected layer, but the model never beats chance accuracy. She suspects each unit in her conv layer only "sees" a 3×3 patch, so the network cannot observe a whole object at once. Help her design a deeper network whose final feature map has a receptive field covering the entire image.

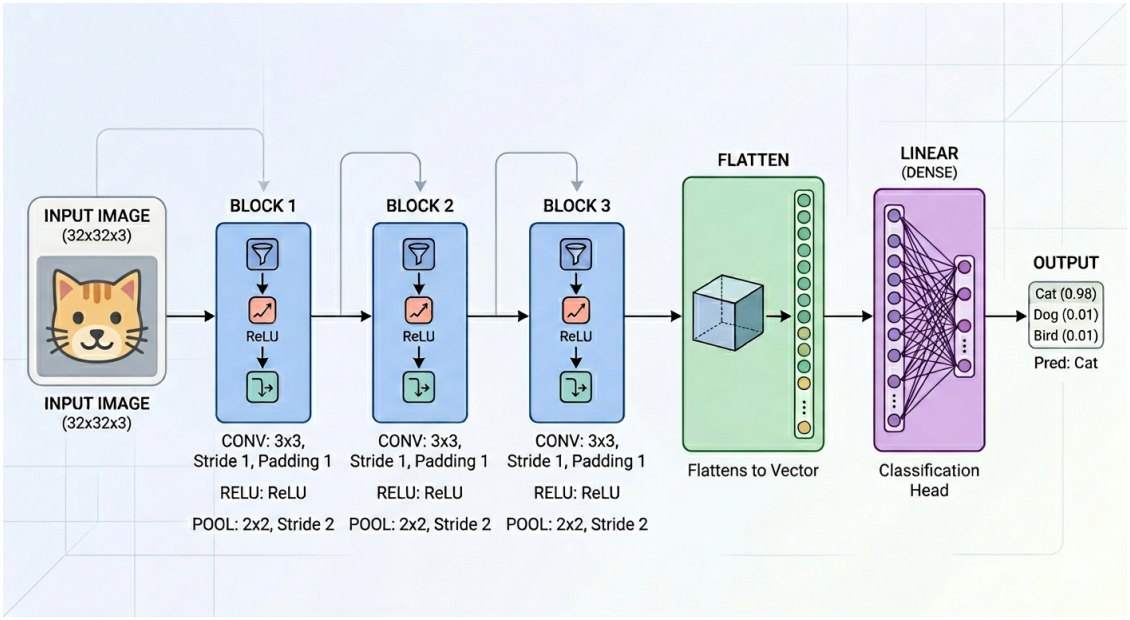


Fig. 6

Network design. Sunny proposes stacking N copies of the following three-layer **Block** on top of the $32 \times 32 \times 3$ input (followed by Flatten and a Linear classifier at the end). Each Block applies, in order:

(i) A convolutional layer with m filters of size 3×3, stride 1, padding 1.

(ii) A ReLU activation.

(iii) A max-pool layer with 2×2 kernel, stride 2, no padding.

1. (2 points) Starting from a 32×32 input, compute the output spatial size after **one** Block, and the receptive field (in input pixels) of a single output unit of that Block.

   Hint: Receptive field of a layer is the number of pixels each output activation of that layer sees.

   SOLUTION:
   Output spatial size: 16×16. Receptive field side length: 4. (The conv outputs each see a 3-wide input window; the 2×2 pool then combines two adjacent conv outputs along each axis, which together span 4 input pixels along each axis.)

2. (6 points) Sunny stacks N Blocks in sequence. Let r_N denote the receptive field side length (in input pixels) of an output activation from the N-th Block. Fill in the table below, then write the equation for r_N as a function of N.

N	spatial size	r_N
1	16×16	4
2	8×8	10
3	4×4	22
4	2×2	46
5	1×1	94

SOLUTION:
$r_N = 3 \cdot 2^N - 2$. (Equivalently, $r_N = 2\, r_{N-1} + 2$ with $r_0 = 1$.)

3. (4 points) What is the minimum number of Blocks N such that $r_N \geq 32$ (i.e., the receptive field covers the entire 32×32 input)? What is the spatial size of the feature map at that N?

SOLUTION:
Solve $3 \cdot 2^N - 2 \geq 32 \Rightarrow 2^N \geq 34/3 \Rightarrow N \geq 4$. So $N = 4$, and the feature map is 2×2.

4. (3 points) Instead of the stack of N Blocks from part 3, Sunny could alternatively use a single convolution with a 32×32 kernel and 32 filters applied to the 3-channel input; this design also has a receptive field covering the entire image (exactly 32). Both designs achieve full-image receptive field. Which design is preferable, and why?

SOLUTION:
The stacked design. Acceptable reasons: fewer learnable parameters; the stack has multiple ReLU non-linearities and can represent nonlinear compositions of pixel features, whereas a single conv is linear in the input; pooling provides partial translation invariance and hierarchical feature learning (edges $\rightarrow$ parts $\rightarrow$ objects); better gradient flow.

5. (3 points) Sunny trains her N-Block stack and observes that training loss is very noisy and unstable across runs with different random seeds. She decides to insert a normalization layer after each convolution (before the ReLU). Due to GPU memory constraints, she must train with **batch size 2**. Should Sunny use Batch Normalization or Layer Normalization? Briefly justify.

SOLUTION:
Layer Normalization. With batch size 2, Batch Normalization's per-batch mean and variance are very noisy estimates and jitter from step to step, destabilizing training. Layer Normalization computes statistics within each example, so it is independent of batch size.

6 Tokenization (8 points) - Recommended 5 Minutes

Please make sure to write your answer only in the provided space.

Consider the following corpus of five sentences:

S1: `"the vibes are immaculate"`
S2: `"u lowkey cooked w/ this assignment"`
S3: `"the assignment is giving main character energy"`
S4: `"this is lowkey the best thing ever"`
S5: `"u cooked the whole thing bestie 🔥"`

Definitions. *Whitespace tokenization* splits text into tokens by breaking on whitespace characters (spaces, tabs, newlines). For example, "the cat sat" becomes ["the", "cat", "sat"].

Byte-Pair Encoding (BPE) is a subword tokenization algorithm that begins by splitting a corpus into individual characters. It then repeatedly identifies the most frequent adjacent pair of symbols and merges it into a new symbol, replacing all occurrences of that pair in the corpus. This process continues until a desired vocabulary size is reached.

1. (2 points) Apply whitespace tokenization to the corpus. Whitespace tokenization means to divide up all the sentences using white spaces. Each word would become a token. How many unique tokens are in the vocabulary?

   SOLUTION:
   Vocabulary (2 pts): Splitting every sentence on spaces and collecting unique tokens gives **21 unique tokens**:

   `the`, `vibes`, `are`, `immaculate`, `u`, `lowkey`, `cooked`, `w/`, `this`, `assignment`, `is`, `giving`, `main`, `character`, `energy`, `best`, `thing`, `ever`, `whole`, `bestie`, 🔥

   also accepted answers: 20 tokens (without the 🔥), and 22 tokens (including whitespace).

2. (6 points) Your friend, in a moment of unhinged genius, trains a BPE tokenizer on a corpus that is just the sentence `"bruh bruh bruh bruh bruh"` repeated 10,000 times. **Assume that whitespace is treated as a separator and, for simplicity, is NOT part of the vocabulary. Whitespaces are also not part of the merge process. Also, (just like the example from lecture) merges cannot cross word boundaries.**

   a) (3 points) What is the final vocabulary after all possible merges?

   b) (3 points) Now suppose your friend uses this tokenizer on the input `"bruhhh"`. Write out the tokens that your BPE tokenizer will produce.

SOLUTION:

Final vocabulary (3 pts):

Since whitespace is not part of the merge process, BPE operates independently on each word. Each occurrence of `bruh` is initially tokenized as:

$$\texttt{b r u h}.$$

Across the corpus, the within-word adjacent pairs `b r`, `r u`, and `u h` each occur 50,000 times. Under any consistent tie-breaking, the merges proceed:

$$\texttt{b} + \texttt{r} \to \texttt{br}, \quad \texttt{br} + \texttt{u} \to \texttt{bru}, \quad \texttt{bru} + \texttt{h} \to \texttt{bruh}.$$

No further merges are possible within a word.

Final vocabulary:

$$\{\texttt{b}, \texttt{r}, \texttt{u}, \texttt{h}, \texttt{br}, \texttt{bru}, \texttt{bruh}\} \quad (\textbf{7 symbols}).$$

Tokenizing `"bruhhh"` (3 pts):

Start from character tokens:

$$\texttt{b r u h h h}.$$

Apply the learned merges in order:

$$\texttt{b r u h h h} \to \texttt{br u h h h} \to \texttt{bru h h h} \to \texttt{bruh h h}.$$

No further merges apply, since neither `bruh+h` nor `h+h` was learned.

$$\boxed{\texttt{bruh h h}}$$

So the final tokenization has 3 tokens.