CSE 493 G1 · SPRING 2026

Midterm Review Recitation

Week of April 21–25 | 3 warm-up MCs + 4 worked problems · 50 min | Midterm: Tuesday April 28

LOGISTICS

Announcements

MIDTERM — TUESDAY APRIL 28

SCOPE

Content up to Lecture 8 (RNN/LSTM won’t be on the midterms but on a quiz soon after)

A2 DUE MONDAY APRIL 27 @ 11:59 PM

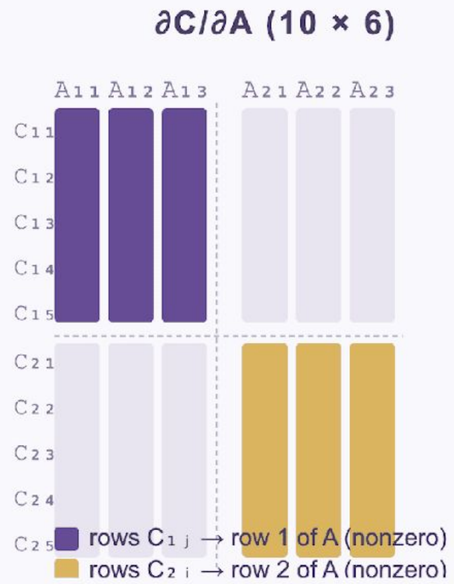
Double sided hand-written cheat sheet will be allowed

Q1 — SOLUTION (A–B)

∂C/∂A: 60 entries, 30 always zero

```
Jacobian shape: (#outputs) × (#inputs)

(a) |C| × |A| = 10 × 6 = 60

──────────────────────────────────────

∂C_ij / ∂A_kl = B_lj if i = k, else 0

C_ij depends ONLY on row i of A

──────────────────────────────────────

(b) Non-zero: 10 × 3 = 30

      Zero: 60 − 30 = 30
```

- For each of the 10 outputs C_ij, only the 3 entries in row i of A have nonzero derivatives — the other row is irrelevant.
- The Jacobian has a **block-diagonal** structure: two



TODAY

Quick poll — what scares you most?

- **A** Backprop / Jacobians
- **B** CNN math (shapes, params)
- **C** RNNs / LSTMs
- **D** BatchNorm / activations

PLAN — 50 MINUTES

- **Warm-up** — 3 MCs from 2025sp
- **Q1** — Jacobians: shapes & zeros
- **Q2** — Backprop + SGD step
- **Q3** — Convolution & pooling
- **Q4** — RNN diagnostics

WARM-UP

Three Quick MCs from 2025 Spring

WARM-UP MC1 · OPTIMIZATION

When using batch GD with a learning rate that is too large, what is the most likely outcome?

A Very slow convergence toward the minimum.

B Divergence or oscillation around the minimum.

C Convergence to the global minimum.

D The gradient norm becomes exactly zero at each step.

E None of the above.

WARM-UP MC1 · SOLUTION

Answer: B — Divergence or oscillation

B Divergence or oscillation around the minimum.

A Very slow convergence toward the minimum.

C Convergence to the global minimum.

D The gradient norm becomes exactly zero.

E None of the above.

1D parabola: $L = \frac{1}{2}w^2$

$w \leftarrow w - \eta \cdot w = (1-\eta) \cdot w$

Stable iff $|1-\eta| < 1$

$\Longrightarrow$ $\eta < 2$

- If η is too large the update overshoots the minimum and the iterate bounces or diverges.
- A would occur with a very *small* lr. C requires the right lr and a convex loss. D never happens for smooth losses.

WARM-UP MC2 · ACTIVITIONS & NORMALIZATION

Mark *all* valid statements.

A ReLU(x) = x if x ≥ 0, else 0.

B Leaky ReLU(x) = max(0.01, x).

C LayerNorm uses batch statistics during inference.

D BatchNorm uses running averages of mean/variance during inference.

WARM-UP MC2 · SOLUTION

Answers: A and D

A ReLU(x) = x if x ≥ 0, else 0. ✓

B Leaky ReLU(x) = max(0.01, x).

C LayerNorm uses batch statistics during inference.

D BatchNorm uses running averages during inference. ✓

B IS WRONG

Leaky ReLU = max(**0.01·x**, x). The 0.01 is a *slope* multiplied by the input — not a constant floor. So negative inputs get a tiny nonzero gradient instead of exactly zero.

C IS WRONG

LayerNorm normalizes each sample across its own features — no batch dimension involved. Behavior is identical at train and test time.

- BatchNorm accumulates running mean and variance during training, and uses those frozen values at test time.

WARM-UP MC3 · WHY CONVOLUTIONS

Key reason to use convolutional layers instead of FC for images?

A Conv layers require more parameters to learn complex patterns.

B Conv layers exploit spatial locality and share weights across locations.

C Conv layers enforce global connectivity between all pixels.

D Conv layers are non-differentiable.

WARM-UP MC3 · SOLUTION

Answer: B — Spatial locality and weight sharing

A Conv requires more parameters.

B Conv exploits spatial locality and shares weights. ✓

C Conv enforces global connectivity.

D Conv is non-differentiable.

3×3 conv, 64→64 channels:

~37K params

Equivalent FC for 224×224 input:

~10^{13} params

- The same filter is applied at every spatial location — that's weight sharing.
- Each output depends only on a local patch — that's spatial locality.
- Together these make deep image models computationally feasible.

PROBLEM
1

Jacobian Dimensions & Zero Structure

Q1

Setup: C = A · B

$$A \in \mathbb{R}^{2\times 3},\ B \in \mathbb{R}^{3\times 5} \Longrightarrow C \in \mathbb{R}^{2\times 5}$$

$$C_{ij} = \Sigma_k\ A_{ik} \cdot B_{kj}$$

QUESTIONS

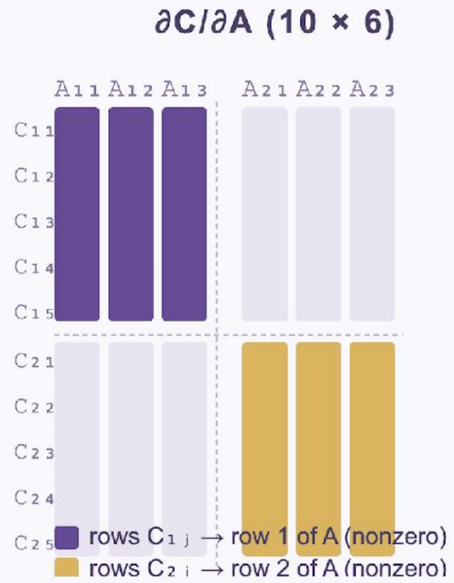
- **(a)** How many total entries in ∂C/∂A?
- **(b)** How many of those are *always* zero, for any A and B?
- **(c)** Repeat (a) for ∂C/∂B.

Q1 — SOLUTION (A–B)

∂C/∂A: 60 entries, 30 always zero

```
Jacobian shape: (#outputs) × (#inputs)

(a) |C| × |A| = 10 × 6 = 60

∂C_ij / ∂A_kl = B_lj if i = k, else 0

C_ij depends ONLY on row i of A

(b) Non-zero: 10 × 3 = 30

     Zero: 60 − 30 = 30
```

- For each of the 10 outputs C_ij, only the 3 entries in row i of A have nonzero derivatives — the other row is irrelevant.
- The Jacobian has a **block-diagonal** structure: two 5×3 blocks, one per row of A.



Q1 — SOLUTION (C)

∂C/∂B: 150 entries, 120 always zero

(c) |C| × |B| = 10 × 15 = **150 entries**

$\partial C_{ij} / \partial B_{kl} = A_{ik}$ if $j = l$, else 0

C_{ij} depends ONLY on column j of B

Non-zero: 10 × 3 = **30**

Zero: 150 − 30 = **120**

Practical backprop (never materializes the Jacobian):

$dL/dA = dL/dC \cdot B^T$

$dL/dB = A^T \cdot dL/dC$

GOOD TO KNOW

The shape rule — Jacobian is (#outputs) × (#inputs) — applies to any function, not just matrix multiplication. Elementwise ops give a diagonal Jacobian; linear ops give a block-diagonal one; add gates give an identity Jacobian.

- The sparsity here (30/60 or 30/150 nonzero) is why backprop avoids building the full tensor — it exploits the structure directly.

PROBLEM
2

Backprop + One SGD Step on a Tiny Network

Q2

2-neuron scalar network

$$z_1 = w_1 \cdot x,\quad a_1 = \mathrm{ReLU}(z_1)$$

$$z_2 = w_2 \cdot x,\quad a_2 = \mathrm{ReLU}(z_2)$$

$$\hat{y} = w_3 \cdot (a_1 + a_2)$$

$$L = \tfrac{1}{2}(\hat{y} - y)^2$$

$x = 1.5 \cdot y = 2 \cdot w_1 = 1.0 \cdot w_2 = -0.5 \cdot w_3 = 3.0$

TASKS

- **(a)** Forward pass: compute z_1, a_1, z_2, a_2, $\hat{y}$, L.
- **(b)** Backward: $\partial L/\partial w_1$, $\partial L/\partial w_2$, $\partial L/\partial w_3$.
- **(c)** One SGD step with lr = 0.1. New weights?
- **(d)** Replace ReLU with identity. How does $\partial L/\partial w_2$ change?

Q2 (A) — FORWARD PASS

Computing all activations

$z_1 = 1.0 \times 1.5 = \mathbf{1.5}$

$a_1 = \max(0, 1.5) = \mathbf{1.5}$

$z_2 = -0.5 \times 1.5 = \mathbf{-0.75}$

$a_2 = \max(0, -0.75) = \mathbf{0}$

$\hat{y} = 3.0 \times (1.5 + 0) = \mathbf{4.5}$

$L = \frac{1}{2} \times (4.5-2)^2 = \mathbf{3.125}$

⚠ $z_2 = -0.75 < 0$

Neuron 2 is DEAD on this input. It contributes nothing to $\hat{y}$, and its gradient will be zero — meaning w_2 gets no update.

Q2 (B) — BACKWARD PASS

Propagating gradients from loss to weights

$\partial L/\partial \hat{y} = \hat{y} - y = 4.5 - 2 = \mathbf{2.5}$

$\partial L/\partial w_3 = 2.5 \times (a_1 + a_2) = 2.5 \times 1.5 = \mathbf{3.75}$

$\partial L/\partial a_1 = 2.5 \times w_3 = \mathbf{7.5}$

$\partial L/\partial z_1 = 7.5 \times 1 = \mathbf{7.5}$ (ReLU active)

$\partial L/\partial w_1 = 7.5 \times 1.5 = \mathbf{11.25}$

$\partial L/\partial z_2 = \square \times \mathbf{0} = \mathbf{0}$ (ReLU dead!)

$\partial L/\partial w_2 = 0 \times x = \mathbf{0}$

ReLU backward = mask.

If the pre-activation $z \leq 0$ in the forward pass, the gradient is zeroed out completely. Everything upstream of a dead neuron receives no training signal.

- The backward pass at each node: multiply upstream gradient by local derivative.
- For ReLU: local derivative is 1 if $z > 0$, else 0.
- Dead ReLU $\Longrightarrow$ w_2 will never update on this input.

Q2 (C–D) — SGD UPDATE & IDENTITY COMPARISON

Updating weights and what changes with identity

(C) SGD — LR = 0.1

w_1: $1.0 - 0.1 \times 11.25 = \mathbf{-0.125}$

w_2: $-0.5 - 0.1 \times 0 = \mathbf{-0.5}$ (unchanged)

w_3: $3.0 - 0.1 \times 3.75 = \mathbf{2.625}$

- w_1 flipped sign — this lr may be too aggressive for this loss landscape.
- w_2 is frozen because its gradient is exactly zero — the dead ReLU blocks all signal.

(D) IDENTITY ACTIVATION → $\partial L/\partial W_2$

$\partial a_2/\partial z_2 = 1$ (identity, not 0)

$\partial L/\partial z_2 = 7.5 \times 1 = \mathbf{7.5}$

$\partial L/\partial w_2 = 7.5 \times 1.5 = \mathbf{11.25}$

- With identity: w_2 would receive a gradient of 11.25 — same as w_1.
- Trade-off: identity avoids dead neurons but loses non-linearity entirely, collapsing the network to a linear function.
- Practical fixes for dead ReLU: Kaiming init, Leaky ReLU, positive bias initialization.

PROBLEM
3

Convolution & Pooling

Q3

4×4 input, 2×2 filter

```
Input I (4×4):
[ 1  0  2  1 ]
[ 2  1  0  0 ]
[ 1  2  1  2 ]
[ 0  1  2  1 ]
```

```
Filter F (2×2):
[ 1  −1 ]
[ 0  1 ]
```

TASKS

- **(a)** Output shape — no padding, stride 1.
- **(b)** Padding = 1, stride = 2. Output dims?
- **(c)** Apply 2×2 maxpool, stride 2 to I.
- **(d)** Strided conv vs. separate pooling — pros and cons.

Q3 (A) — SOLUTION

Output shape: no padding, stride 1

$$H_{out} = \lfloor (H + 2P - F) / S \rfloor + 1$$

$$H_{out} = \lfloor (4 + 0 - 2) / 1 \rfloor + 1$$

$$= \lfloor 2 \rfloor + 1 = \mathbf{3}$$

Output: **3 × 3**

- Sanity check: the top-left corner of F can sit at rows {1,2,3} and cols {1,2,3} — that's 9 valid positions = a 3×3 grid.
- The formula adds 1 because it counts the starting positions of the filter, not the gaps between them.

3×3

output shape

Q3 (B) — SOLUTION

Padding = 1, stride = 2

$$H_{out} = \lfloor (H + 2P - F) / S \rfloor + 1$$

$$H_{out} = \lfloor (4 + 2 \cdot 1 - 2) / 2 \rfloor + 1$$

$$= \lfloor 4/2 \rfloor + 1 = \mathbf{3}$$

Output: **3 × 3** (same size, different reasons)

- Padding = 1 adds a border of zeros — border pixels now participate in more filter windows, preserving edge information.
- Stride = 2 sub-samples — the filter jumps 2 pixels at a time, halving the spatial extent relative to no-stride.
- Classic pattern: **ResNet stem** uses 7×7 conv, stride 2, pad 3 — large receptive field + spatial halving in one step.

3×3

same output, different tradeoffs

Q3 (C) — SOLUTION

Max-pool 2×2, stride 2 on I

INPUT I

1	0	2	1
2	**1**	0	0
1	2	1	2
0	1	2	**1**

4 non-overlapping 2×2 windows

→

OUTPUT

2	**2**
2	**2**

Max-pool = router.

Forward pass: take the maximum of each window and store which position it came from (the argmax).

Backward pass: route the upstream gradient to the argmax position only. All other positions receive gradient 0.

- All four maxima happen to be 2 — a coincidence because I has many 1s and 2s.
- The argmax mask from the forward pass is kept in memory to be used during the backward pass.

Q3 (D) — SOLUTION

Strided conv vs. separate pooling layer

STRIDED CONVOLUTION

- **Pro:** Combines feature extraction and spatial downsampling in a single learnable step. The network can adapt the downsampling to the task.
- **Pro:** Fewer total layers — one operation does two jobs.
- **Con:** The network must simultaneously learn *what* to detect and *where* to subsample, which can be harder to optimize.
- **Con:** Can introduce aliasing if not designed carefully (no anti-aliasing filter).

SEPARATE POOLING LAYER

- **Pro:** Enforces local translation invariance explicitly — nearby shifts in the input produce the same output.
- **Pro:** No parameters; computationally cheap.
- **Con:** Discards spatial information — only the maximum value from each window survives.
- **Con:** Fixed downsampling scheme; the network cannot learn a better one.

Modern practice: **strided conv has largely won** — ResNet uses strided 3×3 convs to downsample. "Why give the network a fixed scheme when it could learn a better one?"

5 Tokenization (20 points) - Recommended 15 Minutes

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

Consider a greedy subword tokenizer processing the sentence:

`"The dog that chased the dog was tired."`

Then, after tokenization, a standard Transformer encoder will receive them as input.

Assume we first run self-attention *without* any positional encodings.

Moreover, assume the sentence is tokenized with a greedy longest-match (maximal-munch) algorithm over a fixed subword vocabulary. At each step, among all vocabulary entries that match the current prefix of the remaining string, choose the longest one (break ties arbitrarily or by smaller ID), emit its ID, remove it from the front, and repeat until the string is consumed.

Example: Suppose the vocabulary contains the tokens {`"he"`:1, `"hello"`:2, `"llo"`:3}. Then the input `"hello"` is tokenized as [2] ("hello") rather than [1,3] ("he","llo"), because the tokenizer greedily matches the longest possible subword.

Given this setup, answer the following questions:

1. (4 points) Suppose we use a greedy subword tokenizer (BPE) with the following dictionary (token → index):

   {`" "` : 1, `"The "` : 2, `"dog "` : 3, `"dog"` : 4, `"that"` : 5,, `"chased"` : 6, `"the"` : 7,

   `"dog."` : 8, `"was"` : 9, `"tired"` : 10, `"."` : 11}.

   Describe step by step how this sentence will be tokenized using the longest-match rule, and list the resulting token indices.

(a) "The " → 2. Remaining: "dog that chased the dog was tired."

(b) "dog " → 3. Remaining: "that chased the dog was tired."

(c) "that" → 5. Remaining: " chased the dog was tired."

(d) " " → 1. Remaining: "chased the dog was tired."

(e) "chased" → 6. Remaining: " the dog was tired."

(f) " " → 1. Remaining: "the dog was tired."

(g) "the" → 7. Remaining: " dog was tired."

(h) " " → 1. Remaining: "dog was tired."

(i) "dog " → 3. Remaining: "was tired."

(j) "was" → 9. Remaining: " tired."

(k) " " → 1. Remaining: "tired."

(l) "tired" → 10. Remaining: "."

(m) "." → 11. Remaining: ""

Resulting token indices: $[2, 3, 5, 1, 6, 1, 7, 1, 3, 9, 1, 10, 11]$.

2. (4 points) Given the tokenization you came up with from the last question as input, how will the model differentiate between the two occurrences of the token "dog" when no positional encodings are used? Provide a brief conceptual explanation.

2. (4 points) Given the tokenization you came up with from the last question as input, how will the model differentiate between the two occurrences of the token "dog" when no positional encodings are used? Provide a brief conceptual explanation.

Without positional encodings, each "dog" token has the same embedding, so its query, key, and value vectors are identical. Self-attention uses only these vectors, so both positions compute the same attention weights and produce the same output.

3. (4 points) Suppose we use a greedy subword tokenizer (BPE) with a new dictionary (token → index), note that only index 4 is changed from "dog" to " dog" compared to the previous dictionary:

$$\{\texttt{" "}: 1,\ \texttt{"The "}: 2,\ \texttt{"dog "}: 3,\ \texttt{" dog"}: 4,\ \texttt{"that"}: 5,,\ \texttt{"chased"}: 6,\ \texttt{"the"}: 7,$$

$$\texttt{"dog."}: 8,\ \texttt{"was"}: 9,\ \texttt{"tired"}: 10,\ \texttt{"."}: 11\}.$$

Describe step by step how this sentence will be tokenized using the longest-match rule, and list the resulting token indices.

(h) " dog " $\rightarrow$ 4. Remaining: "was tired."

(i) "was" $\rightarrow$ 9. Remaining: " tired."

(j) " " $\rightarrow$ 1. Remaining: "tired."

(k) "tired" $\rightarrow$ 10. Remaining: "."

(l) "." $\rightarrow$ 11. Remaining: ""

Resulting token indices: $[2, 3, 5, 1, 6, 1, 7, 4, 9, 1, 10, 11]$.

4. (4 points) Given the tokenization you came up with from the last question as input, how will the model differentiate between the two occurrences of the token "dog" when no positional encodings are used? Provide a brief conceptual explanation.

(h) " dog " → 4. Remaining: "was tired."

(i) "was" → 9. Remaining: " tired."

(j) " " → 1. Remaining: "tired."

(k) "tired" → 10. Remaining: "."

(l) "." → 11. Remaining: ""

Resulting token indices: $[2, 3, 5, 1, 6, 1, 7, 4, 9, 1, 10, 11]$.

4. (4 points) Given the tokenization you came up with from the last question as input, how will the model differentiate between the two occurrences of the token "dog" when no positional encodings are used? Provide a brief conceptual explanation.

Although without positional encodings, the two "dog" tokens are encoded into different embeddings ("dog " and " dog"), so their query, key, and value vectors are different. Self-attention will then produce different outputs, which means that the model is able to differentiate the two.

5. (4 points) Explain why positional encodings remain crucial for capturing sequence order and structural relationships in a Transformer, even when token-level embeddings differ between occurrences.

5. (4 points) Explain why positional encodings remain crucial for capturing sequence order and structural relationships in a Transformer, even when token-level embeddings differ between occurrences.

5. (4 points) Explain why positional encodings remain crucial for capturing sequence order and structural relationships in a Transformer, even when token-level embeddings differ between occurrences.

Transformers are inherently permutation-invariant: without positional encodings, they cannot infer token order or relative distances. Positional encodings inject absolute (and via sinusoidals, relative) position information into embeddings, enabling the model to track sequence structure, learn n-gram patterns, and capture long-range dependencies that depend on token positions.