New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6923: Written Homework 3.
Due Friday, Dec. 5th, 2025, 11:59pm.

Discussion with other students is allowed for this problem set, but solutions must be written-up individually.

10% *extra credit will be given if solutions are typewritten (using LaTeX, Markdown, or another mathematical formatting program). MSWord with Equation Editor does not count.*

New Policy: *While we allow you to use AI models while solving the problems, you should (1) mention in your solution that you used AI, and (2) write your final solutions down yourself. If we suspect that you copied your solution directly from AI, we will set up a meeting with you where you will need to explain your solution. If you fail to do so, you will receive zero points for the homework.*

Problem 1: Convolution as a Linear Layer (20 points)

Consider a 2D convolutional layer with c_o output channels, kernel size (k, k), and stride s, applied to a single input image of size (h, w, c_i), where c_i is the number of input channels. (We ignore the batch dimension throughout.) The parameters of the convolutional layer are given by

$$W \in \mathbb{R}^{c_o \times c_i \times k \times k},$$

and we assume there is no bias for simplicity.

Assume there is no padding, and that the stride s divides both $(h - k)$ and $(w - k)$ so that the output spatial dimensions are integers.

(a) In class, we discussed how a convolution is a special case of a linear layer. Describe the general linear layer (as in `torch.nn.Linear`) that implements the convolution above. Specifically, explain:

- What are the dimensions of the input and output to this linear layer?
- How should the input image be transformed before being passed to this linear layer?
- What is the shape of the weight matrix for this linear layer?
- How are the weights of this linear layer structured in relation to the convolutional parameters W?

(b) Now view this convolutional layer as a linear map $y = Ax$ with a weight matrix

$$A \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}},$$

where d_{in} and d_{out} are the input and output dimensions you described in part (a).

Compare this to a general *dense* linear layer with the same input and output dimensions (i.e., an arbitrary A with no structure). What is the fraction of *entries of the full matrix* A that are fixed to zero by the convolutional structure (i.e., the effective sparsity)? Express your answer in terms of h, w, c_i, c_o, k, and s.

Hint: For part (a), the output spatial dimensions of the convolution are

$$h' = \frac{h-k}{s} + 1, \qquad w' = \frac{w-k}{s} + 1.$$

Solution to Problem 1: Convolution as a Linear Layer

(a) Representing a convolution as a linear layer. We consider a single input image $X \in \mathbb{R}^{h \times w \times c_i}$ (ignoring the batch dimension) and a convolutional layer with

$$W \in \mathbb{R}^{c_o \times c_i \times k \times k}, \quad \text{stride } s, \quad \text{no padding, no bias.}$$

By assumption, s divides both $(h-k)$ and $(w-k)$, so the output spatial dimensions are

$$h' = \frac{h-k}{s} + 1, \qquad w' = \frac{w-k}{s} + 1,$$

and the convolutional output tensor has shape

$$Y \in \mathbb{R}^{h' \times w' \times c_o}.$$

We now describe the corresponding `torch.nn.Linear`-style layer.

Dimensions of the input and output.

- The input image X is flattened into a vector

$$x \in \mathbb{R}^{d_{\text{in}}}, \quad \text{where } d_{\text{in}} = h \cdot w \cdot c_i.$$

- The output Y is flattened into a vector

$$y \in \mathbb{R}^{d_{\text{out}}}, \quad \text{where } d_{\text{out}} = h' \cdot w' \cdot c_o.$$

- Thus the corresponding linear layer has input dimension d_{in} and output dimension d_{out}.

How to transform the input image.

We fix an ordering of the entries of X to obtain $x \in \mathbb{R}^{hwc_i}$, for example:

$$x = \text{vec}(X) = \begin{bmatrix} X_{1,1,1} \\ X_{1,1,2} \\ \vdots \\ X_{1,1,c_i} \\ X_{1,2,1} \\ \vdots \\ X_{h,w,c_i} \end{bmatrix}.$$

That is, we can iterate over spatial positions (row, column) and, for each position, iterate over channels. Any fixed, consistent ordering is acceptable.

Similarly, we flatten the output tensor Y into

$$y = \text{vec}(Y) \in \mathbb{R}^{h'w'c_o},$$

for example stacking over spatial locations and then channels.

Shape of the weight matrix.

A standard linear layer $y = Ax$ has a weight matrix

$$A \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}} = \mathbb{R}^{(h'w'c_o) \times (hwc_i)}.$$

Structure of the weight matrix in terms of W.

Each entry of Y is produced by applying one of the c_o convolutional filters to a $k \times k$ patch of the input across all c_i channels.

Let us index:

- Output spatial positions by (u, v), where $u = 0, \dots, h' - 1$ and $v = 0, \dots, w' - 1$.
- Output channels by $o = 1, \dots, c_o$.
- Input spatial positions by (i, j), where $i = 0, \dots, h - 1$ and $j = 0, \dots, w - 1$.
- Input channels by $c = 1, \dots, c_i$.

For a given output position (u, v) and channel o, the convolution computes

$$Y_{u,v,o} = \sum_{c=1}^{c_i} \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} W_{o,c,a,b} \cdot X_{us+a,\ vs+b,\ c}.$$

In the linear representation $y = Ax$, $Y_{u,v,o}$ corresponds to a particular component y_r for some row index r (determined by our flattening convention). The corresponding row $A_{r,:}$ has:

- Non-zero entries exactly at the columns corresponding to the input indices

$$(i, j, c) = (us + a,\ vs + b,\ c), \quad \text{for } a, b \in \{0, \dots, k - 1\},\ c \in \{1, \dots, c_i\}.$$

- The value at the column corresponding to $(us + a, vs + b, c)$ is

$$A_{r,\mathrm{col}(us+a,vs+b,c)} = W_{o,c,a,b}.$$

- All other entries in that row are zero.

Thus each row of A has exactly $k^2 c_i$ non-zero entries, corresponding to one receptive field, and these non-zero values are copies of the filter coefficients in W. The full matrix A is therefore extremely sparse and has a structured, "shifted" pattern of non-zero blocks that reflects the sliding convolution across the image.

(b) Fraction of entries fixed to zero (effective sparsity). We now compare:

- A general dense linear layer with input dimension $d_{\text{in}} = hwc_i$ and output dimension $d_{\text{out}} = h'w'c_o$.
- The special linear layer with weight matrix $A \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ that corresponds to the convolution above.

A general dense linear layer has

$$\#\text{entries}_{\text{dense}} = d_{\text{out}} \cdot d_{\text{in}} = (h'w'c_o) \cdot (hwc_i).$$

The linear operator induced by the convolution can still be written as a matrix $A \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, but most of its entries are fixed to zero.

Number of non-zero entries (connections) in A.

Each output entry $Y_{u,v,o}$ (and hence each component of y) depends on exactly $k^2 c_i$ entries of X. Therefore each row of A has exactly $k^2 c_i$ non-zero entries, and there are $d_{\text{out}} = h'w'c_o$ rows. Hence:

$$\#\text{non-zero entries in } A = d_{\text{out}} \cdot (k^2 c_i) = (h'w'c_o) \cdot (k^2 c_i).$$

Fraction of entries fixed to zero.

The total number of entries in A is

$$\#\text{entries in } A = d_{\text{out}} \cdot d_{\text{in}} = (h'w'c_o) \cdot (hwc_i).$$

Thus the fraction of *non-zero* entries is

$$\text{density} = \frac{\#\text{non-zero}}{\#\text{total}} = \frac{(h'w'c_o)(k^2 c_i)}{(h'w'c_o)(hwc_i)} = \frac{k^2}{hw}.$$

Therefore, the fraction of entries that are fixed to zero (the effective sparsity) is

$$\boxed{\text{sparsity} = 1 - \frac{k^2}{hw}.}$$

Note that this fraction is independent of c_i, c_o, and the stride s: these quantities cancel when comparing the number of non-zero entries to the total number of entries of A. The stride s affects h' and w' (and thus the size of A), but not the ratio of non-zero to total entries.

Remark. This sparsity calculation counts entries of the full matrix A that are fixed to zero. In addition, many non-zero entries of A are tied together (weight sharing): all spatial locations for a given output channel reuse the same $k^2 c_i$ filter values from W. Thus the number of *free* trainable parameters is just

$$\#\text{trainable conv parameters} = c_o \cdot c_i \cdot k^2,$$

which is usually much smaller than both the total number of entries and the number of non-zero entries in A.

Problem 2: Sequence Modeling with Transformers and (30 points)

In this problem, we consider an autoregressive sequence modeling setup. We observe the sequence

$$u_1,\ u_2,\ u_1,\ u_2,$$

where u_1 and u_2 are drawn independently from $\mathrm{Unif}([0,1])$ (e.g., using `torch.rand`).

Autoregressive convention. We will assume that the input sequence is $x = (0, u_1, u_2, u_1)$ and the desired output sequence is $y = (u_1, u_2, u_1, u_2)$. At position i, the model receives only the prefix $(x_1, \ldots, x_i)$ as input and must produce a prediction o_i for $y_i = x_{i+1}$ (where for convenience we use $x_5 = y_4 = u_2$). In particular, the model does *not* observe x_{i+1} before emitting our prediction for it o_i.

Specifically,

The loss for a prediction sequence (o_1, o_2, o_3, o_4) is

$$L = \sum_{i=1}^{4} |o_i - y_i|.$$

We are interested in the lowest possible *expected* loss

$$\mathbb{E}_{u_1,u_2\sim\mathrm{Unif}([0,1])}\left[\sum_{i=1}^{4} |o_i - y_i|\right].$$

(a) Consider any possible model (not necessarily implementable by a neural network). What is the lowest possible expected loss

$$\mathbb{E}_{u_1,u_2}\left[\sum_{i=1}^{4} |o_i - y_i|\right]?$$

What predictions must an optimal model make at each position?

(b) Now suppose we use an RNN whose hidden state is a single `float32` number. There is no embedding layer: the RNN receives x_{i-1} and the previous hidden state h_{i-1}, and outputs a new state h_i and the prediction o_i.

Can such an RNN achieve the optimal expected loss from part (a)? Justify your answer.

(c) Now consider a transformer consisting of a *single causal self-attention layer*, with:

- no layer normalization,
- no skip connections,
- no MLP blocks,
- a single attention head,
- query, key, and value vectors all of dimension 1,
- the attention output *is* the model prediction o_i (i.e., no separate readout layer),
- the causal attention mask ensures that token i may only attend to keys from positions $1, \ldots, i$.

Each input at position i consists of (p_i, x_i), where p_i is a positional encoding vector that *you* may choose.

Design:

$$W_Q,\ W_K,\ W_V,\ p_1, p_2, p_3, p_4$$

so that this transformer approximately achieves the optimal loss from part (a).

Solution to Problem 2: Sequence Modeling with Transformers and RNNs

We observe a sequence

$$u_1, u_2, u_1, u_2,$$

with $u_1, u_2 \sim \mathrm{Unif}([0,1])$ i.i.d.

The input and target sequences are

$$x = (x_1, \dots, x_4) = (0, u_1, u_2, u_1), \quad y = (y_1, \dots, y_4) = (u_1, u_2, u_1, u_2).$$

At position i, the model has only the prefix $(x_1, \dots, x_i)$ and must output o_i as a prediction of $y_i = x_{i+1}$ (with $x_5 := u_2$).

The loss is

$$L = \sum_{i=1}^{4} |o_i - y_i|,$$

and we are interested in the optimal expected loss

$$\mathbb{E}_{u_1,u_2 \sim \mathrm{Unif}([0,1])}\left[\sum_{i=1}^{4} |o_i - y_i|\right].$$

(a) Optimal predictions with an arbitrary model. For each position i, an optimal predictor under absolute loss $|o_i - y_i|$ is a (measurable) function of the available information that outputs a conditional *median* of y_i given the prefix $(x_1, \dots, x_i)$.

We analyze each time step:

- **Position 1.** The model has only seen $x_1 = 0$. The target is

  $$y_1 = u_1 \sim \mathrm{Unif}([0,1]),$$

  independent of anything observed so far. The optimal constant under L_1 is the median, so the optimal prediction is

  $$o_1^\star = \tfrac{1}{2}.$$

  The optimal expected absolute error is

  $$\mathbb{E}|u_1 - \tfrac{1}{2}| = \int_0^1 |u - \tfrac{1}{2}|\, du = 2\int_0^{1/2} (\tfrac{1}{2} - u)\, du = 2\left[\tfrac{1}{2}u - \tfrac{1}{2}u^2\right]_0^{1/2} = \tfrac{1}{4}.$$

- **Position 2.** Now the model has seen $x_1 = 0$ and $x_2 = u_1$. The target is

  $$y_2 = u_2,$$

  where $u_2 \sim \mathrm{Unif}([0,1])$ is independent of u_1. Thus the conditional distribution of y_2 given (x_1, x_2) is still $\mathrm{Unif}([0,1])$. Again the optimal L_1 predictor is

  $$o_2^\star = \tfrac{1}{2},$$

  with

  $$\mathbb{E}|u_2 - \tfrac{1}{2}| = \tfrac{1}{4}.$$

- **Position 3.** The model has seen the entire prefix

  $$(x_1, x_2, x_3) = (0, u_1, u_2).$$

  The target is

  $$y_3 = u_1.$$

  At this point both u_1 and u_2 are known exactly from the prefix, so the optimal predictor can simply output

  $$o_3^\star = u_1,$$

  yielding zero loss at this position:

  $$|o_3^\star - y_3| = |u_1 - u_1| = 0.$$

- **Position 4.** The model now has seen $x_1, \dots, x_4 = (0, u_1, u_2, u_1)$, so again it knows u_1 and u_2 exactly. The target is
$$y_4 = u_2.$$
The optimal predictor is
$$o_4^\star = u_2,$$
with zero loss:
$$|o_4^\star - y_4| = |u_2 - u_2| = 0.$$

Therefore the minimal possible expected loss is

$$\mathbb{E}\left[\sum_{i=1}^{4} |o_i^\star - y_i|\right] = \mathbb{E}|u_1 - \tfrac{1}{2}| + \mathbb{E}|u_2 - \tfrac{1}{2}| + 0 + 0 = \tfrac{1}{4} + \tfrac{1}{4} = \boxed{\tfrac{1}{2}}.$$

An optimal model (not necessarily a neural network) can achieve this loss by making the following predictions:

$$o_1 = \tfrac{1}{2}, \quad o_2 = \tfrac{1}{2}, \quad o_3 = u_1, \quad o_4 = u_2.$$

(b) Can a 1D `float32` RNN achieve the optimal loss? From part (a), the optimal expected loss is $\frac{1}{2}$. This comes entirely from positions 1 and 2; at positions 3 and 4 we can, in principle, predict perfectly:

$$o_3 = u_1, \quad o_4 = u_2,$$

so that $|o_3 - y_3| = |o_4 - y_4| = 0$.

For an RNN with a 1-dimensional `float32` hidden state to achieve this optimal loss, it must also have *zero* error at positions 3 and 4 for (almost) all (u_1, u_2).

Let h_3 be the hidden state after processing the prefix $(x_1, x_2, x_3) = (0, u_1, u_2)$. To predict perfectly, the RNN must be able to:

- recover u_1 in order to produce $o_3 = u_1$ using the information in h_3 (and possibly x_3), and
- later recover u_2 in order to produce $o_4 = u_2$ from (h_3, x_4), where $x_4 = u_1$.

In other words, from h_3 we must be able to reconstruct *both* u_1 and u_2:

$$(u_1, u_2) \quad \longmapsto \quad h_3$$

must be injective (one-to-one), because otherwise there would be two different pairs $(u_1, u_2) \neq (u_1', u_2')$ that give the same h_3, and then the RNN could not output the correct (u_1, u_2) for both sequences at positions 3 and 4.

However, all of these numbers are stored as `float32`. There are:

- at most 2^{32} possible values of the hidden state h_3, but
- about $(2^{32})^2 = 2^{64}$ possible pairs (u_1, u_2).

By the pigeonhole principle, the mapping $(u_1, u_2) \mapsto h_3$ cannot be injective: there must be distinct pairs (u_1, u_2) and (u_1', u_2') that produce the same h_3.

For such colliding pairs, the RNN (which only "remembers" h_3 and the current input) cannot output the correct (u_1, u_2) for both sequences at positions 3 and 4. Thus it must make some positive error on those positions for at least one of the sequences, and therefore the *expected* loss at positions 3 and 4 is strictly positive.

Since the optimal total expected loss is $\frac{1}{2}$ and this already accounts for the unavoidable error at positions 1 and 2, any additional positive error at positions 3 and 4 means the RNN's expected loss must be strictly larger than $\frac{1}{2}$.

It is fine if the students don't make the very formal counting argument, but they need to argue that we cannot encode two `float32` numbers exactly in one `float32` number.

$$\boxed{\text{A 1D float32 RNN cannot achieve the optimal expected loss from part (a).}}$$

(c) Single-layer causal self-attention achieving (approximately) optimal loss. We now consider a single causal self-attention layer with:

- no layer normalization,
- no skip connections,
- no MLP blocks,
- a single attention head,
- query, key, and value *dimension* $d_k = d_v = 1$,
- the attention output at position i is directly the model prediction o_i.

Each input token at position i is

$$z_i = (p_i, x_i),$$

where:

- x_i is the scalar sequence value $(0, u_1, u_2, u_1)$,
- $p_i \in \mathbb{R}^4$ is a positional encoding.

We choose **one-hot positional encodings**:

$$p_1 = e_1 = (1,0,0,0)^\top, \quad p_2 = e_2 = (0,1,0,0)^\top, \quad p_3 = e_3 = (0,0,1,0)^\top, \quad p_4 = e_4 = (0,0,0,1)^\top.$$

So $z_i \in \mathbb{R}^5$ is the concatenation $z_i = (p_i, x_i)$.

We use standard scaled dot-product attention with causal masking:

$$q_i = W_Q z_i \in \mathbb{R}, \; k_i = W_K z_i \in \mathbb{R}, \; v_i = W_V z_i \in \mathbb{R},$$

$$\alpha_{ij} = \begin{cases} \dfrac{\exp(q_i k_j)}{\sum_{\ell=1}^{i} \exp(q_i k_\ell)} & j \le i, \\ 0 & j > i, \end{cases}$$

and

$$o_i = \sum_{j=1}^{i} \alpha_{ij} v_j.$$

We will design W_Q, W_K, W_V so that:

$$o_1 \approx \tfrac{1}{2}, \quad o_2 \approx \tfrac{1}{2}, \quad o_3 \approx u_1, \quad o_4 \approx u_2.$$

Values v_j

We want:

- $v_1 = 0.5$ (a constant "memory token" for the first two positions),
- $v_2 = x_2 = u_1$,
- $v_3 = x_3 = u_2$,
- v_4 arbitrary (we will make position 4 ignore it).

We choose

$$W_V = \begin{bmatrix} 0.5 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Then, for $z_i = (p_i, x_i)$, we have

$$v_i = 0.5 \cdot (p_i)_1 + 1 \cdot x_i.$$

Explicitly:

$$\begin{aligned} v_1 &= 0.5 \cdot 1 + 1 \cdot x_1 = 0.5 + 1 \cdot 0 = 0.5, \\ v_2 &= 0.5 \cdot 0 + 1 \cdot x_2 = u_1, \\ v_3 &= 0.5 \cdot 0 + 1 \cdot x_3 = u_2, \\ v_4 &= 0.5 \cdot 0 + 1 \cdot x_4 = u_1. \end{aligned}$$

We want the attention pattern (with causal masking):

- At position 1: only token 1 is visible, so $o_1 = v_1 = 0.5$ automatically.
- At position 2: attend *almost entirely* to token 1 $\Rightarrow o_2 \approx v_1 = 0.5$.
- At position 3: attend *almost entirely* to token 2 $\Rightarrow o_3 \approx v_2 = u_1$.
- At position 4: attend *almost entirely* to token 3 $\Rightarrow o_4 \approx v_3 = u_2$.

Because p_i are one-hot, we can choose W_Q, W_K so that queries and keys depend only on the position, not on x_i. Let $M > 0$ be a large scaling constant (we use it to sharpen the softmax). Define

$$W_Q = \begin{bmatrix} 0 & M & -M & M & 0 \end{bmatrix}, \quad W_K = \begin{bmatrix} M & 0 & 2M & -M & 0 \end{bmatrix}.$$

Then for $i = 1, \ldots, 4$:

$$\begin{aligned} q_1 &= W_Q z_1 = 0 \cdot 1 + M \cdot 0 - M \cdot 0 + M \cdot 0 + 0 \cdot x_1 = 0, \\ q_2 &= W_Q z_2 = 0 \cdot 0 + M \cdot 1 - M \cdot 0 + M \cdot 0 + 0 \cdot x_2 = M, \\ q_3 &= W_Q z_3 = 0 \cdot 0 + M \cdot 0 - M \cdot 1 + M \cdot 0 + 0 \cdot x_3 = -M, \\ q_4 &= W_Q z_4 = 0 \cdot 0 + M \cdot 0 - M \cdot 0 + M \cdot 1 + 0 \cdot x_4 = M, \end{aligned}$$

and

$$\begin{aligned} k_1 &= W_K z_1 = M \cdot 1 + 0 \cdot 0 + 2M \cdot 0 - M \cdot 0 + 0 \cdot x_1 = M, \\ k_2 &= W_K z_2 = M \cdot 0 + 0 \cdot 1 + 2M \cdot 0 - M \cdot 0 + 0 \cdot x_2 = 0, \\ k_3 &= W_K z_3 = M \cdot 0 + 0 \cdot 0 + 2M \cdot 1 - M \cdot 0 + 0 \cdot x_3 = 2M, \\ k_4 &= W_K z_4 = M \cdot 0 + 0 \cdot 0 + 2M \cdot 0 - M \cdot 1 + 0 \cdot x_4 = -M. \end{aligned}$$

The unnormalized attention scores are $s_{ij} = q_i k_j$ (scaled by $\sqrt{d_k} = 1$). For each position:

- **Position 1** $(i = 1)$**.** The causal mask only allows $j = 1$. Thus $\alpha_{11} = 1$, and
$$o_1 = v_1 = 0.5.$$
- **Position 2** $(i = 2)$**.** Only $j = 1, 2$ are allowed. With $q_2 = M$, $k_1 = M$, $k_2 = 0$, we have
$$s_{21} = q_2 k_1 = M^2, \quad s_{22} = q_2 k_2 = 0.$$
For large M,
$$\alpha_{21} = \frac{e^{M^2}}{e^{M^2} + e^0} \approx 1, \quad \alpha_{22} \approx 0.$$
Therefore
$$o_2 \approx v_1 = 0.5.$$
- **Position 3** $(i = 3)$**.** Allowed $j = 1, 2, 3$. With $q_3 = -M$, we get
$$s_{31} = q_3 k_1 = -M^2, \quad s_{32} = q_3 k_2 = 0, \quad s_{33} = q_3 k_3 = -2M^2.$$
So
$$\alpha_{32} = \frac{e^0}{e^{-M^2} + e^0 + e^{-2M^2}} \approx 1, \quad \alpha_{31}, \alpha_{33} \approx 0.$$
Hence
$$o_3 \approx v_2 = u_1.$$

- **Position 4 ($i = 4$).** Allowed $j = 1, 2, 3, 4$. With $q_4 = M$, we have

$$s_{41} = q_4 k_1 = M^2, \quad s_{42} = q_4 k_2 = 0, \quad s_{43} = q_4 k_3 = 2M^2, \quad s_{44} = q_4 k_4 = -M^2.$$

  For large M, the largest score is s_{43}, so

$$\alpha_{43} \approx 1, \quad \alpha_{41}, \alpha_{42}, \alpha_{44} \approx 0.$$

  Thus

$$o_4 \approx v_3 = u_2.$$

Putting this together, as $M \to \infty$ the attention weights become arbitrarily close to one-hot in the desired pattern, and the outputs satisfy

$$o_1 \approx \tfrac{1}{2}, \quad o_2 \approx \tfrac{1}{2}, \quad o_3 \approx u_1, \quad o_4 \approx u_2.$$

Therefore the expected loss of this transformer can be made arbitrarily close to the optimal value $\frac{1}{2}$ by taking M sufficiently large.

$$\boxed{\text{The single-head, 1D transformer with these } W_Q, W_K, W_V, p_i \text{ achieves (approximately) the optimal expected loss.}}$$