New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6923: Written Homework 3.
Due Friday, April 24th, 2026, 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

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

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).