ECE/SIOC 228: Machine Learning for Physical Applications Homework - 2

Yanxi Chen, A69029717

Time Spent: 10 hours

1 Written Questions [60 Pts]

1.1 Neural Network Basics: Forward Propagation, Loss, and Backpropagation [18 pts]

Q1.1: Given the basic feedforward neural network architecture:

- Vector $x_i \in \mathbb{R}^3$ in the input
- Weight matrix $W \in \mathbb{R}^{2\times 3}$, bias vector $b \in \mathbb{R}^2$ in the hidden layer pre-activation
- Vector $z \in \mathbb{R}^2$ and $h \in \mathbb{R}^2$ in the hidden layer activation
- Weight vector $v \in \mathbb{R}^2$, bias scalar $c \in \mathbb{R}$ in the output layer

The computation of the output $\hat{y}_i$ for a single input x_i is as follows:

$$z = Wx_i + b,$$

where Wx_i is a matrix-vector multiplication resulting in $z \in \mathbb{R}^2$.

$$h = \text{ReLU}(z) = \max(0, z),$$

applied element-wise to z, giving $h \in \mathbb{R}^2$.

$$\hat{y}_i = v^\top h + c$$

This is a dot product between v and h plus scalar bias c, yielding $\hat{y}_i \in \mathbb{R}$.

Q1.2: Full loss expression:

$$L(W, b, v, c) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \left(v^\top \text{ReLU}(Wx_i + b) + c\right)\right)^2$$

ReLU effect: Makes the loss function piecewise linear with respect to W and b, makes Wx_i + b non-negative, and creating non-differentiable points where any $z_j = 0$.

Q1.3: For a single data point (x, y), with $L = \frac{1}{2}(y - \hat{y})^2$: For a single sample (x, y), we define:

$$\delta = \hat{y} - y, \quad z = Wx + b, \quad h = \text{ReLU}(z), \quad \hat{y} = v^\top h + c.$$

Because

$$\frac{\partial L}{\partial \hat{y}} = \delta$$

Then

$$\frac{\partial L}{\partial v} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial v} = \delta\, h, \quad \frac{\partial L}{\partial c} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial c} = \delta,$$

and since

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h} = \delta\, v, \quad \frac{d}{dz}\text{ReLU}(z) = \mathbf{1}_{z>0},$$

we get

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial h}\frac{dh}{dz} = (\delta\, v)\odot\mathbf{1}_{z>0}, \quad \frac{\partial L}{\partial W} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial W} = \Big((\delta\, v)\odot\mathbf{1}_{z>0}\Big)\, x^\top, \quad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial b} = (\delta\, v)\odot\mathbf{1}_{z>0},$$

where $\odot$ represents element-wise product.

Q1.4: Initial parameters and computed gradients for $(x, y) = \left(\begin{bmatrix}1\\0\\2\end{bmatrix}, 3\right)$:

$$\begin{aligned}
W^{(0)} &= \begin{bmatrix}1 & -1 & 0\\ 0 & 2 & 1\end{bmatrix}, \quad \frac{\partial L}{\partial W} = \begin{bmatrix}-8 & 0 & -16\\ 4 & 0 & 8\end{bmatrix}\\
b^{(0)} &= \begin{bmatrix}0\\1\end{bmatrix}, \quad \frac{\partial L}{\partial b} = \begin{bmatrix}-8\\4\end{bmatrix}\\
v^{(0)} &= \begin{bmatrix}2\\-1\end{bmatrix}, \quad \frac{\partial L}{\partial v} = \begin{bmatrix}-4\\-12\end{bmatrix}\\
c^{(0)} &= 0, \quad \frac{\partial L}{\partial c} = -4\\
\eta &= 0.02
\end{aligned}$$

Gradient descent update with learning rate η:

1. **Weight Matrix W update**:

$$\begin{aligned}
W^{(1)} &= W^{(0)} - \eta\frac{\partial L}{\partial W},\\
W^{(1)} &= \begin{bmatrix}1 & -1 & 0\\ 0 & 2 & 1\end{bmatrix} - 0.02\begin{bmatrix}-8 & 0 & -16\\ 4 & 0 & 8\end{bmatrix}\\
&= \begin{bmatrix}1 & -1 & 0\\ 0 & 2 & 1\end{bmatrix} + \begin{bmatrix}0.16 & 0 & 0.32\\ -0.08 & 0 & -0.16\end{bmatrix}\\
&= \begin{bmatrix}1.16 & -1 & 0.32\\ -0.08 & 2 & 0.84\end{bmatrix}
\end{aligned}$$

2. **Bias Vector b update**:

$$\begin{aligned} b^{(1)} &= b^{(0)} - \eta \frac{\partial L}{\partial b}, \\ b^{(1)} &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} - 0.02 \begin{bmatrix} -8 \\ 4 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.16 \\ -0.08 \end{bmatrix} \\ &= \begin{bmatrix} 0.16 \\ 0.92 \end{bmatrix} \end{aligned}$$

3. **Output Weight v update**:

$$\begin{aligned} v^{(1)} &= v^{(0)} - \eta \frac{\partial L}{\partial v}, \\ v^{(1)} &= \begin{bmatrix} 2 \\ -1 \end{bmatrix} - 0.02 \begin{bmatrix} -4 \\ -12 \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ -1 \end{bmatrix} + \begin{bmatrix} 0.08 \\ 0.24 \end{bmatrix} \\ &= \begin{bmatrix} 2.08 \\ -0.76 \end{bmatrix} \end{aligned}$$

4. **Output Bias c update**:

$$\begin{aligned} c^{(1)} &= c^{(0)} - \eta \frac{\partial L}{\partial c}, \\ c^{(1)} &= 0 - 0.02 \times (-4) = 0.08 \end{aligned}$$

Compute $\hat{y}^{(1)}$ with updated parameters:

$$
\begin{aligned}
z^{(1)} &= W^{(1)}x + b^{(1)} \\
&= \begin{bmatrix} 1.16 & -1 & 0.32 \\ -0.08 & 2 & 0.84 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.16 \\ 0.92 \end{bmatrix} \\
&= \begin{bmatrix} 1.16 \cdot 1 + (-1) \cdot 0 + 0.32 \cdot 2 \\ (-0.08) \cdot 1 + 2 \cdot 0 + 0.84 \cdot 2 \end{bmatrix} + \begin{bmatrix} 0.16 \\ 0.92 \end{bmatrix} \\
&= \begin{bmatrix} 1.16 + 0 + 0.64 \\ -0.08 + 0 + 1.68 \end{bmatrix} + \begin{bmatrix} 0.16 \\ 0.92 \end{bmatrix} \\
&= \begin{bmatrix} 1.8 \\ 1.6 \end{bmatrix} + \begin{bmatrix} 0.16 \\ 0.92 \end{bmatrix} = \begin{bmatrix} 1.96 \\ 2.52 \end{bmatrix} \\
h^{(1)} &= \mathrm{ReLU}(z^{(1)}) = \begin{bmatrix} 1.96 \\ 2.52 \end{bmatrix} \quad (\text{both } > 0) \\
\hat{y}^{(1)} &= {v^{(1)}}^{\top} h^{(1)} + c^{(1)} \\
&= \begin{bmatrix} 2.08 & -0.76 \end{bmatrix} \begin{bmatrix} 1.96 \\ 2.52 \end{bmatrix} + 0.08 \\
&= 2.08 \times 1.96 + (-0.76) \times 2.52 + 0.08 \\
&= 4.0768 - 1.9152 + 0.08 = 2.2416 \\
L^{(0)} &= 8 \\
L^{(1)} &= \frac{1}{2}(3 - 2.2416)^2 = \frac{1}{2}(0.7584)^2 \approx 0.2876
\end{aligned}
$$

The loss decreased from 8 to 0.2876, showing the effectiveness of the gradient descent.

The choice of learning rate: Too large η causes oscillation, too small leads to slow convergence. Optimal choice balances convergence speed and stability.

1.2 Universal Approximation Theorem for Neural Networks [8 pts]

Q2.2: Let $m = 1$. To approximate the piecewise constant function:

$$
f(x) = \begin{cases} 0, & 0 \le x < 0.5, \\ 1, & 0.5 \le x \le 1, \end{cases}
$$

choose

$$
w_1 = M, \quad b_1 = -M \cdot 0.5, \quad v_1 = 1, \quad c = 0,
$$

with $M \to \infty$. Using the property of Sigmoid function:

$$
\hat{f}(x) = \sigma\big(M(x - 0.5)\big) \xrightarrow[M \to \infty]{} \begin{cases} 0, & x < 0.5, \\ 1, & x > 0.5. \end{cases}
$$

Q2.3: Let $m = 2$. To approximate

$$f(x) = \begin{cases} 0, & 0 \le x < 1, \\ 1, & 1 \le x \le 2, \\ 3, & 2 \le x \le 3, \end{cases}$$

use two "steps" at $x = 1$ and $x = 2$ using two neurons:

$$\begin{aligned} w_1 = M, \quad b_1 = -M \cdot 1, \quad v_1 = 1, \\ w_2 = M, \quad b_2 = -M \cdot 2, \quad v_2 = 2, \quad c = 0, \end{aligned}$$

When $x < 1$:

$$\begin{gathered} M(x-1) \xrightarrow[M\to\infty]{} -\infty, \quad M(x-2) \xrightarrow[M\to\infty]{} -\infty, \\ \sigma\big(M(x-1)\big) \xrightarrow[M\to\infty]{} 0, \quad \sigma\big(M(x-2)\big) \xrightarrow[M\to\infty]{} 0, \\ \therefore \quad \hat{f}(x) = 1 \cdot 0 + 2 \cdot 0 = 0. \end{gathered}$$

When $1 \le x \le 2$:

$$\begin{gathered} M(x-1) \xrightarrow[M\to\infty]{} +\infty, \quad M(x-2) \xrightarrow[M\to\infty]{} -\infty, \\ \sigma\big(M(x-1)\big) \xrightarrow[M\to\infty]{} 1, \quad \sigma\big(M(x-2)\big) \xrightarrow[M\to\infty]{} 0, \\ \therefore \quad \hat{f}(x) = 1 \cdot 1 + 2 \cdot 0 = 1. \end{gathered}$$

When $x > 2$:

$$\begin{gathered} M(x-1) \xrightarrow[M\to\infty]{} +\infty, \quad M(x-2) \xrightarrow[M\to\infty]{} +\infty, \\ \sigma\big(M(x-1)\big) \xrightarrow[M\to\infty]{} 1, \quad \sigma\big(M(x-2)\big) \xrightarrow[M\to\infty]{} 1, \\ \therefore \quad \hat{f}(x) = 1 \cdot 1 + 2 \cdot 1 = 3. \end{gathered}$$

So that

$$\hat{f}(x) = \sigma\big(M(x-1)\big) + 2\,\sigma\big(M(x-2)\big) \xrightarrow[M\to\infty]{} \begin{cases} 0, & x < 1, \\ 1, & 1 \le x \le 2, \\ 3, & x > 2. \end{cases}$$

Q2.4: Since the Universal Approximation Theorem states that "a feedforward neural network with a single hidden layer and enough neurons can approximate any continuous function on a compact subset of to arbitrary accuracy. We can think of any continuous function as a sum of many tiny step-like pieces, and because sigmoids can closely mimic those steps, adding enough sigmoids lets the network trace the function as precisely as we like.

1.3 Transformer basics: Single-head attention [15pts]

Given an input sequence $X = [x_1; x_2; x_3] \in \mathbb{R}^{3\times d}$ and weight matrices $W_Q, W_K, W_V \in \mathbb{R}^{d\times d_k}$. The goal is to compute

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big) V.$$

Q3.1: The expressions for Q, K, V in terms of X and W_Q, W_K, W_V:

$$Q = X\,W_Q, \quad K = X\,W_K, \quad V = X\,W_V,$$

where $X \in \mathbb{R}^{3\times d}$ and each $W \in \mathbb{R}^{d\times d_k}$ gives

$$Q, K, V \in \mathbb{R}^{3\times d_k}.$$

The attention score matrix S

$$S = Q\,K^\top \in \mathbb{R}^{3\times 3},$$

since Q is $3 \times d_k$ and $K^\top$ is $d_k \times 3$.

Q3.2: Let $d = d_k = 2$ and

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad W_Q = W_K = W_V = I_2.$$

Then

$$Q = K = V = X \in \mathbb{R}^{3\times 2}.$$

Compute

$$S = QK^\top = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}.$$

We have $d_k = 2$, so

$$\widetilde{S} = \frac{S}{\sqrt{2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{2}{\sqrt{2}} \end{bmatrix}$$

Define $\alpha_{ij} = \exp\big(\widetilde{S}_{ij}\big)$. We have,

$$\exp\left(\frac{1}{\sqrt{2}}\right) \approx 2.028, \quad \exp(0) = 1, \quad \exp\Big(\sqrt{2}\Big) \approx 4.113.$$

Row 1: $(\alpha_{11}, \alpha_{12}, \alpha_{13}) = [2.028,\ 1,\ 2.028], \quad \sum = 5.056,$

$\text{softmax}(\widetilde{S})_1 = \big[2.028/5.056,\ 1/5.056,\ 2.028/5.056\big] \approx [0.401,\ 0.198,\ 0.401].$

Row 2: $(\alpha_{21}, \alpha_{22}, \alpha_{23}) = [1,\ 2.028,\ 2.028], \quad \sum = 5.056,$

$\text{softmax}(\widetilde{S})_2 \approx [0.401,\ 0.198,\ 0.401].$

Row 3: $(\alpha_{31}, \alpha_{32}, \alpha_{33}) = [2.028,\ 2.028,\ 4.113], \quad \sum = 8.169,$

$\text{softmax}(\widetilde{S})_3 \approx [0.248,\ 0.248,\ 0.503].$

Finally, since $V = X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$,

$$Y = \text{softmax}(\widetilde{S})\, V \approx \begin{bmatrix} 0.401 & 0.198 & 0.401 \\ 0.198 & 0.401 & 0.401 \\ 0.248 & 0.248 & 0.503 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.802 & 0.599 \\ 0.599 & 0.802 \\ 0.751 & 0.751 \end{bmatrix}.$$

Q3.3: If each component of $q_i, k_j \in \mathbb{R}^{d_k}$ is an independent random variable with mean 0 and variance 1, then

$$\text{Var}(q_i \cdot k_j) = \sum_{\ell=1}^{d_k} \text{Var}(q_{i\ell} k_{j\ell}) = d_k \cdot (1 \cdot 1) = d_k.$$

Thus the magnitude of $q_i \cdot k_j$ grows like its standard deviation $\sqrt{d_k}$. Dividing by $\sqrt{d_k}$ keeps the inputs to softmax in a range with non-vanishing gradients, avoiding extremely small gradients when d_k is large.

Q3.4: With $X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, we get

$$H = \text{ReLU}(XW_1 + b_1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix},$$

$$Y_{\text{MLP}} = H\, W_2 + b_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

In contrast, single-head attention (from Q3.2) produced

$$Y_{\text{att}} \approx \begin{bmatrix} 0.802 & 0.599 \\ 0.599 & 0.802 \\ 0.751 & 0.751 \end{bmatrix},$$

showing that the MLP leaves each row equal to the input (when using identity weights), whereas attention mixes rows together into new combinations.

Q3.5:

(a) **Mixing input information:** Self-attention dynamically weights and mixes tokens based on their pairwise similarity, so each output token pulls information from whichever other tokens are most relevant. An MLP uses the same fixed weights at every position, processing each token independently.

(b) **Capturing relationships:** Because self-attention explicitly compares every token to every other token, it can directly capture long-range and context-dependent relationships. As for MLP, with its position-wise operations, cannot adaptively link distant tokens and thus cannot handle with such interactions.

1.4 Transformer basics: Multi-head attention [11pts]

Q4.1:

Head 1:

$$W_Q^{(1)} = W_K^{(1)} = W_V^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 2}.$$

Thus:

$$Q^{(1)} = XW_Q^{(1)} = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 1 \end{bmatrix},$$

Similarly, $K^{(1)} = Q^{(1)}$, $V^{(1)} = Q^{(1)}$. All are 3×2.

Head 2:

$$W_Q^{(2)} = W_K^{(2)} = W_V^{(2)} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 2}.$$

Then

$$Q^{(2)} = XW_Q^{(2)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad K^{(2)} = Q^{(2)}, \quad V^{(2)} = Q^{(2)},$$

each of size 3×2.

Q4.2: Head 1:

$$S^{(1)} = Q^{(1)}K^{(1)\top} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 2 & 3 \\ 2 & 1 & 1 \\ 3 & 1 & 2 \end{bmatrix}.$$

Scaling by $1/\sqrt{2} \approx 0.7071$ gives

$$\widetilde{S}^{(1)} \approx \begin{bmatrix} 3.5355 & 1.4142 & 2.1213 \\ 1.4142 & 0.7071 & 0.7071 \\ 2.1213 & 0.7071 & 1.4142 \end{bmatrix}$$

.

Combine the softmax of each row of S:

$$A^{(1)} = \text{softmax}(\widetilde{S}^{(1)}) \approx \begin{bmatrix} 0.734 & 0.088 & 0.178 \\ 0.503 & 0.248 & 0.248 \\ 0.576 & 0.140 & 0.284 \end{bmatrix}.$$

Head 2:

$$S^{(2)} = Q^{(2)} K^{(2)\top} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix},$$

$$A^{(2)} = \text{softmax}(\widetilde{S}^{(2)}) \approx \begin{bmatrix} 0.401 & 0.198 & 0.401 \\ 0.198 & 0.401 & 0.401 \\ 0.248 & 0.248 & 0.503 \end{bmatrix}.$$

Explain briefly: Watching head's projection matrices: Head 1 attends based on the first two features of each token (it "looks at" dimensions 1–2), while head 2 attends based on the last two features (dimensions 3–4).

Q4.3:

Compute:

$$Y^{(1)} = A^{(1)} V^{(1)}, Y^{(2)} = A^{(2)} V^{(2)}.$$

We get:

$$Y^{(1)} \approx \begin{bmatrix} 0.912 & 1.734 \\ 0.753 & 1.502 \\ 0.860 & 1.576 \end{bmatrix}, \quad Y^{(2)} \approx \begin{bmatrix} 0.599 & 0.802 \\ 0.802 & 0.599 \\ 0.751 & 0.751 \end{bmatrix}.$$

Concatenating along the feature dimension gives

$$Y = [\, Y^{(1)} \,\big|\, Y^{(2)} \,] = \begin{bmatrix} 0.912 & 1.734 & 0.599 & 0.802 \\ 0.753 & 1.502 & 0.802 & 0.599 \\ 0.860 & 1.576 & 0.751 & 0.751 \end{bmatrix} \in \mathbb{R}^{3\times 4},$$

and since $W_O = I_4$, this is the final output.

Q4.4:

- (a) Splitting dimensions across heads lets each head specialize on different subspaces of the input—for example, one head can focus on coarse, high-variance features while another captures fine, low-variance patterns.
- (b) If heads aren't coordinated or trained properly, they can duplicate each other or learn conflicting views, potentially harming overall performance.

1.5 Transformer basics: Positional Encoding [8pts]

Q5.1: Here $d = 4$, so i range from 0 to $\frac{d}{2} - 1 = 1$

$$10000^{2i/4} = \begin{cases} 10000^0 = 1, & i = 0, \\ 10000^{1/2} = 100, & i = 1. \end{cases}$$

Thus for each *pos*:

$$\mathrm{PE}(pos) = \left[\sin\tfrac{pos}{1},\ \cos\tfrac{pos}{1},\ \sin\tfrac{pos}{100},\ \cos\tfrac{pos}{100}\right].$$

$pos = 0$: $\sin 0 = 0,\ \cos 0 = 1$:

$$\mathrm{PE}(0) = [\,0,\ 1,\ 0,\ 1\,].$$

$pos = 1$:

$$\mathrm{PE}(1) = [\,\sin 1,\ \cos 1,\ \sin 0.01,\ \cos 0.01\,].$$

$pos = 2$:

$$\mathrm{PE}(2) = [\,\sin 2,\ \cos 2,\ \sin 0.02,\ \cos 0.02\,].$$

Q5.2:

(1) The period T_i for the function $\sin\big(pos/10000^{2i/d}\big)$ satisfies $\frac{pos+T_i}{10000^{2i/d}} = \frac{p}{10000^{2i/d}} + 2\pi$, so

$$T_i = 2\pi \cdot 10000^{2i/d}.$$

For $d = 4$:

$$T_0 = 2\pi \cdot 1 \approx 6.28, \quad T_1 = 2\pi \cdot 100 \approx 628.3.$$

(2) Thus dimension $i = 0$ oscillates quickly (period ≈ 6 positions), while $i = 1$ oscillates slowly (period ≈ 628 positions). Lower-i channels capture high-frequency position changes; higher-i channels capture low-frequency position patterns.

(3) For fixed i, $\mathrm{PE}(p, i)$ is a pure sine or cosine in p, so it is infinitely differentiable (smooth) and repeats exactly every T_i steps (periodic).

Q5.3:

- (a) Adding positional encoding to each embedding row $X_{pos} \in \mathbb{R}^d$ gives

$$\widetilde{X}_{pos} = X_{pos} + \mathrm{PE}(pos)\,.$$

- (b) Because

$$\widetilde{X}_p \cdot \widetilde{X}_q = X_p \cdot X_q + X_p \cdot \mathrm{PE}(q) + X_q \cdot \mathrm{PE}(p) + \mathrm{PE}(p) \cdot \mathrm{PE}(q),$$

the last term $\mathrm{PE}(p) \cdot \mathrm{PE}(q)$ using dot product and we can get things like $sinpsinq + cospcosq = cos(p - q)$, which depends on phase difference $|p - q|$. Thus the model can capture relative position information from those sinusoidal phase differences.