Homework 1: Backpropagation - Theory Part

Rishabh Patil (rbp5812@nyu.edu)

September, 2025

1 Theory

1.1 Two-Layer Neural Nets

The architecture is given as: Input $x \in \mathbb{R}^n \to \text{Linear}_1 \to f \to \text{Linear}_2 \to g \to \hat{y} \in \mathbb{R}^K$
Where:

- $\text{Linear}_i(x) = W^{(i)}x + b^{(i)}$ is an affine transformation.
- f and g are element-wise non-linear activation functions.

1.2 Regression Task

- Activation functions: $f(\cdot) = 5\text{ReLU}(\cdot)$ and $g(\cdot)$ is the identity function.
- Loss function: $\ell_{\text{MSE}}(\hat{y}, y) = \|\hat{y} - y\|^2$, where y is the target output.

(a) Programming Steps for Training with PyTorch using SGD on a Single Batch

The five programming steps to train this model with PyTorch using SGD on a single batch of data are:

1. **Forward Pass:**
   - Define the model architecture (including Linear layers, ReLU, and identity).
   - Pass the input data x through the model to obtain the predicted output $\hat{y}$:
   $$\hat{y} = g\Big(\mathbf{W}^{(2)} f\big(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big) + \mathbf{b}^{(2)}\Big).$$
   - Since $g(\cdot)$ is the identity function, this simplifies to
   $$\hat{y} = \mathbf{W}^{(2)} f\big(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big) + \mathbf{b}^{(2)}.$$
2. **Compute the Loss:**
   - Calculate the loss using the Mean Squared Error (MSE) loss function:
   $$\ell_{\text{MSE}}(\hat{y}, y) = \|\hat{y} - y\|^2 = \sum_{i=1}^{K} (\hat{y}_i - y_i)^2.$$
3. **Zero Gradients:**
   - Before computing new gradients, clear any existing gradients in the model's parameters (weights and biases). In PyTorch, this is typically done with `model.zero_grad()`.
   - Mathematically, this corresponds to resetting the accumulated gradients to zero:
   $$G_{\mathbf{W}^{(1)}} = 0, \quad G_{\mathbf{b}^{(1)}} = 0, \quad G_{\mathbf{W}^{(2)}} = 0, \quad G_{\mathbf{b}^{(2)}} = 0.$$

4. **Backward Pass (Backpropagation):**

   - Compute the gradients of the loss with respect to the model's parameters using the `L.backward()` method on the loss tensor. This automatically computes gradients through the defined computation graph.
   - Mathematically, this corresponds to assigning each accumulated gradient as follows:

$$G_{\mathbf{W}^{(2)}} \leftarrow \frac{\partial \ell}{\partial \mathbf{W}^{(2)}}, \quad G_{\mathbf{b}^{(2)}} \leftarrow \frac{\partial \ell}{\partial \mathbf{b}^{(2)}}, \quad G_{\mathbf{W}^{(1)}} \leftarrow \frac{\partial \ell}{\partial \mathbf{W}^{(1)}}, \quad G_{\mathbf{b}^{(1)}} \leftarrow \frac{\partial \ell}{\partial \mathbf{b}^{(1)}}.$$

5. **Optimizer Step (Gradient Update):**

   - Use an optimizer (e.g., SGD) to update the model's parameters based on the computed gradients. This involves subtracting the learning rate multiplied by the gradient from the current parameter value. In PyTorch, this is typically done with `optimizer.step()`.
   - The optimizer's `step()` method directly modifies the parameters of the model. For SGD, this would be an update like:

$$W^{(i)} \leftarrow W^{(i)} - \eta \frac{\partial \ell}{\partial W^{(i)}}$$

$$b^{(i)} \leftarrow b^{(i)} - \eta \frac{\partial \ell}{\partial b^{(i)}}$$

     where η is the learning rate.

(b) Forward Pass Inputs and Outputs for a Single Data Point (x, y)

Let $x \in \mathbb{R}^n$ be the input and $y \in \mathbb{R}^K$ be the target output.

Linear 1:

- Input: $x \in \mathbb{R}^n$
- Output: $z_1 = W^{(1)}x + b^{(1)}$
  - $W^{(1)} \in \mathbb{R}^{h \times n}$
  - $b^{(1)} \in \mathbb{R}^h$
  - $z_1 \in \mathbb{R}^h$ (where h is the number of neurons in the first hidden layer)

Activation f ($5 \cdot$ReLU(z_1)):

- Input: $z_1 \in \mathbb{R}^h$
- Output: $z_2 = f(z_1) = 5 \cdot \mathrm{ReLU}(z_1)$
  - $z_2 \in \mathbb{R}^h$

Linear 2:

- Input: $z_2 \in \mathbb{R}^h$
- Output: $z_3 = W^{(2)}z_2 + b^{(2)}$
  - $W^{(2)} \in \mathbb{R}^{K \times h}$
  - $b^{(2)} \in \mathbb{R}^K$
  - $z_3 \in \mathbb{R}^K$

Activation g (Identity):

- Input: $z_3 \in \mathbb{R}^K$
- Output: $\hat{y} = g(z_3) = z_3$
  - $\hat{y} \in \mathbb{R}^K$

(c) Gradients Calculated from the Backward Pass

We need to compute the gradients of the loss $\ell = \ell_{\text{MSE}}(\hat{y}, y)$ with respect to the parameters $W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$.

Gradient w.r.t. $W^{(2)}$ and $b^{(2)}$:

- The output layer transformation involves $\hat{y} = g(z_3)$. Given g is the identity function, $\hat{y} = z_3$. Thus,

$$\frac{\partial \ell}{\partial \mathbf{z_3}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}$$

  .

- The linear transformation for z_3 is $z_3 = W^{(2)} z_2 + b^{(2)}$, where $W^{(2)} \in \mathbb{R}^{K \times h}$, $z_2 \in \mathbb{R}^{h \times 1}$, $b^{(2)} \in \mathbb{R}^{K \times 1}$, and $z_3 \in \mathbb{R}^{K \times 1}$
- Using matrix calculus rules, the derivative of z_3 with respect to $W^{(2)}$ is:

$$\frac{\partial z_3}{\partial \mathbf{W^{(2)}}} = z_2^T$$

  This results in a row vector of size $1 \times h$.

- Applying the chain rule to find $\frac{\partial \ell}{\partial W^{(2)}}$:

$$\frac{\partial \ell}{\partial W^{(2)}} = \left(\frac{\partial \ell}{\partial z_3} \right) \left(\frac{\partial z_3}{\partial \mathbf{W^{(2)}}} \right)^T$$

  Substituting $\frac{\partial \ell}{\partial \mathbf{z_3}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}$ and $\frac{\partial z_3}{\partial \mathbf{W^{(2)}}} = z_2^T$:

$$\frac{\partial \ell}{\partial \mathbf{W^{(2)}}} = \left(\frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} \right) (z_2^T)^T$$

$$\boxed{\frac{\partial \ell}{\partial \mathbf{W}^{(2)}} = \left(\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}_3} \right) \boldsymbol{z}_2^\top}$$

  This outer product results in a matrix of size $K \times h$.

- For the bias term $b^{(2)}$:

$$\frac{\partial \ell}{\partial \mathbf{b^{(2)}}} = \frac{\partial \ell}{\partial \mathbf{z_3}} \frac{\partial \mathbf{z_3}}{\partial \mathbf{b^{(2)}}}$$

  Since $z_3 = W^{(2)} z_2 + b^{(2)}$, $\frac{\partial \mathbf{z_3}}{\partial \mathbf{b^{(2)}}} = I_K$ (the identity matrix).

$$\frac{\partial \ell}{\partial \mathbf{b^{(2)}}} = \left(\frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} \right) I_K$$

$$\boxed{\frac{\partial \ell}{\partial \mathbf{b^{(2)}}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}}$$

  This gradient is a column vector of size $K \times 1$.

Gradient w.r.t. z_2:

- The gradient of the loss with respect to z_2 is computed using the chain rule:

$$\frac{\partial \ell}{\partial \mathbf{z_2}} = \left(\frac{\partial \mathbf{z_3}}{\partial \mathbf{z_2}}\right)^T \frac{\partial \ell}{\partial \mathbf{z_3}}$$

- Given $z_3 = W^{(2)} z_2 + b^{(2)}$, the Jacobian

$$\frac{\partial \mathbf{z_3}}{\partial \mathbf{z_2}} = \mathbf{W^{(2)}}$$

.

- The term $\frac{\partial \ell}{\partial \mathbf{z_3}}$ is given by

$$\frac{\partial \ell}{\partial \mathbf{z_3}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}$$

.

- Substituting these into the chain rule formula:

$$\frac{\partial \ell}{\partial \mathbf{z_2}} = (\mathbf{W^{(2)}})^T \left(\frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}\right)$$

This gradient $\frac{\partial \ell}{\partial \mathbf{z_2}}$ is a column vector of size $h \times 1$.

Gradient w.r.t. z_1 (and propagating back):

- The relationship between z_1 and z_2 is $z_2 = f(z_1)$.
- The gradient of the loss with respect to z_2 is $\frac{\partial \ell}{\partial \mathbf{z_2}} = (\mathbf{W^{(2)}})^T \left(\frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}\right)$, as derived in Paragraph 2.
- Using the chain rule for element-wise operations, the gradient of the loss with respect to z_1 is:

$$\frac{\partial \ell}{\partial \mathbf{z_1}} = \frac{\partial \ell}{\partial \mathbf{z_2}} \frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}$$

- Substituting the expression for $\frac{\partial \ell}{\partial \mathbf{z_2}}$:

$$\frac{\partial \ell}{\partial \mathbf{z_1}} = \left((\mathbf{W^{(2)}})^T \left(\frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}\right)\right) \left(\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}\right)$$

This gradient $\frac{\partial \ell}{\partial \mathbf{z_1}}$ is a column vector of dimensions $h \times 1$.

Gradient w.r.t. $W^{(1)}$ and $b^{(1)}$:

- The first layer's linear transformation is $z_1 = W^{(1)} x + b^{(1)}$.
- Using matrix calculus rules, the derivative of z_1 with respect to $W^{(1)}$ is:

$$\frac{\partial \mathbf{z_1}}{\partial \mathbf{W^{(1)}}} = x^T$$

- **Gradient w.r.t. $W^{(1)}$:** Applying the chain rule and matrix calculus rules:

$$\frac{\partial \ell}{\partial \mathbf{W^{(1)}}} = \frac{\partial \ell}{\partial \mathbf{z_1}} \frac{\partial \mathbf{z_1}}{\partial \mathbf{W^{(1)}}}$$

$$\frac{\partial \ell}{\partial \mathbf{W^{(1)}}} = \left(\frac{\partial \ell}{\partial \mathbf{z_1}}\right) x^T$$

Substituting the expression for $\frac{\partial \ell}{\partial \mathbf{z_1}}$:

$$\boxed{\frac{\partial \ell}{\partial \mathbf{W^{(1)}}} = \left(\left((\mathbf{W^{(2)}})^T \left(\frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}\right)\right) \left(\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}\right)\right) x^T}$$

This gradient matrix has dimensions $h \times n$.

- For the bias term $b^{(1)}$:

$$\frac{\partial \ell}{\partial \mathbf{b^{(1)}}} = \frac{\partial \ell}{\partial \mathbf{z_1}} \frac{\partial \mathbf{z_1}}{\partial \mathbf{b^{(1)}}}$$

Since $z_1 = W^{(1)}x + b^{(1)}$, $\frac{\partial \mathbf{z_1}}{\partial \mathbf{b^{(1)}}} = I_K$ (the identity matrix).

$$\frac{\partial \ell}{\partial \mathbf{b^{(1)}}} = \left(\frac{\partial \ell}{\partial \mathbf{z_1}}\right) I_K$$

$$\frac{\partial \ell}{\partial \mathbf{b^{(1)}}} = \frac{\partial \ell}{\partial \mathbf{z_1}}$$

Substituting the expression for $\frac{\partial \ell}{\partial \mathbf{z_1}}$:

$$\boxed{\frac{\partial \ell}{\partial \mathbf{b^{(1)}}} = \left((\mathbf{W^{(2)}})^T \left(\frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}\right)\right) \left(\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}\right)}$$

This gradient vector has dimensions $h \times 1$.

(d) Elements of $\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}$, $\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}$, and $\frac{\partial \ell}{\partial \hat{\mathbf{y}}}$

Given:

- $z_1 \in \mathbb{R}^h$ (input to the first hidden layer activation)
- $z_2 \in \mathbb{R}^h$ (output of the first hidden layer activation)
- $\hat{y} \in \mathbb{R}^K$ (network's predicted output)
- $y \in \mathbb{R}^K$ (true target output)

Elements of $\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}$:

- The problem states that the output layer activation function $g(\cdot)$ is the identity function. Therefore, $\hat{y} = g(z_3) = z_3$.
- The Jacobian is defined as

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial z_{3,1}} & \cdots & \frac{\partial \hat{y}_1}{\partial z_{3,K}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_K}{\partial z_{3,1}} & \cdots & \frac{\partial \hat{y}_K}{\partial z_{3,K}} \end{bmatrix}.$$

- Since $\hat{y}_i = z_{3,i}$, each element satisfies

$$\frac{\partial \hat{\mathbf{y}}_\mathbf{i}}{\partial \mathbf{z_{3,j}}} = \delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j, \end{cases}$$

where δ_{ij} is the Kronecker delta.

- This describes the $K \times K$ identity matrix.
- Thus, the Jacobian matrix is

$$\boxed{\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} = I_K \in \mathbb{R}^{K \times K}.}$$

Elements of $\frac{\partial z_2}{\partial z_1}$:

- The first hidden layer activation function is defined as $f(\cdot) = 5 \cdot \text{ReLU}(\cdot)$. Thus, $\boldsymbol{z}_2 = f(\boldsymbol{z}_1) = 5 \cdot \text{ReLU}(\boldsymbol{z}_1)$.
- The Jacobian is defined as
$$\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{z}_1} = \begin{bmatrix} \frac{\partial z_{2,1}}{\partial z_{1,1}} & \cdots & \frac{\partial z_{2,1}}{\partial z_{1,m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_{2,m}}{\partial z_{1,1}} & \cdots & \frac{\partial z_{2,m}}{\partial z_{1,m}} \end{bmatrix}.$$
- The ReLU function is $\text{ReLU}(u) = \max(0, u)$. Its derivative is
$$\frac{d}{du}\text{ReLU}(u) = \begin{cases} 1 & \text{if } u > 0, \\ 0 & \text{if } u \leq 0. \end{cases}$$
- Since $\boldsymbol{z}_2 = 5 \cdot \text{ReLU}(\boldsymbol{z}_1)$ is applied element-wise, the Jacobian is diagonal. The derivative of $z_{2,i}$ w.r.t. $z_{1,j}$ is non-zero only when $i = j$.
- For $i = j$:
$$\frac{\partial z_{2,i}}{\partial z_{1,i}} = \frac{\partial}{\partial z_{1,i}}\big(5 \cdot \text{ReLU}(z_{1,i})\big) = 5 \cdot \frac{d}{du}\text{ReLU}(u)\Big|_{u=z_{1,i}} = \begin{cases} 5 & \text{if } z_{1,i} > 0, \\ 0 & \text{if } z_{1,i} \leq 0. \end{cases}$$
- Thus, the elements of the Jacobian are
$$\boxed{\left(\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{z}_1}\right)_{ij} = \begin{cases} 5 & \text{if } i = j \text{ and } z_{1,i} > 0, \\ 0 & \text{otherwise.} \end{cases}}$$
- This matrix is diagonal and encodes the slope of the scaled ReLU at each coordinate.
- **Dimensionality:** $m \times m$.

Elements of $\frac{\partial \ell}{\partial \hat{y}}$:

- The loss function is Mean Squared Error (MSE): $\ell(\hat{y}, y) = \|\hat{y} - y\|^2 = \sum_{i=1}^{K}(\hat{y}_i - y_i)^2$.
- To find the gradient of the loss with respect to the predicted output vector $\hat{y}$, we compute the partial derivative of ℓ with respect to each element $\hat{y}_i$:
$$\frac{\partial \ell}{\partial \hat{\mathbf{y}}_\mathbf{i}} = \frac{\partial}{\partial \hat{\mathbf{y}}_\mathbf{i}}\left((\hat{y}_i - y_i)^2 + \sum_{j \neq i}(\hat{y}_j - y_j)^2\right)$$
$$\frac{\partial \ell}{\partial \hat{\mathbf{y}}_\mathbf{i}} = 2(\hat{y}_i - y_i)$$
- **Dimensionality:** $\hat{y}, y \in \mathbb{R}^K$, $\frac{\partial \ell}{\partial \hat{y}} \in \mathbb{R}^K$.
- In expanded vector form, the gradient is
$$\frac{\partial \ell}{\partial \hat{\mathbf{y}}} = \begin{bmatrix} 2(\hat{y}_1 - y_1) \\ 2(\hat{y}_2 - y_2) \\ \vdots \\ 2(\hat{y}_K - y_K) \end{bmatrix}.$$
- Equivalently, in compact notation:
$$\boxed{\frac{\partial \ell}{\partial \hat{\mathbf{y}}} = 2(\hat{y} - y).}$$

1.3 Classification Task

We would like to perform multi-class classification task, so we set $f = \tanh$ and $g = \sigma$, the logistic sigmoid function $\sigma(z) = (1 + \exp(-z))^{-1}$.

(a) Changes for MSE Loss

Changes in Equations for Forward Pass (Part 1.2 (b)):

- The activation function f would change from 5ReLU$(\cdot)$ to $\tanh(\cdot)$.
- The activation function g would change from identity to $\sigma(\cdot)$.
- The output layer would be $\hat{y} = \sigma(z_3)$, where $z_3 = W^{(2)}z_2 + b^{(2)}$ and $z_2 = \tanh(z_1)$.

If we use the same MSE loss function $\ell_{\text{MSE}}(\hat{y}, y) = \|\hat{y} - y\|^2$:
Linear 1:

- Input: $x \in \mathbb{R}^n$
- Output: $z_1 = W^{(1)}x + b^{(1)}$
  - $W^{(1)} \in \mathbb{R}^{h \times n}$
  - $b^{(1)} \in \mathbb{R}^h$
  - $z_1 \in \mathbb{R}^h$ (where h is the number of neurons in the first hidden layer)

Activation f ($\tanh(\cdot)$):

- Input: $z_1 \in \mathbb{R}^h$
- Output: $z_2 = f(z_1) = \tanh(z_1)$
  - $z_2 \in \mathbb{R}^h$

Linear 2:

- Input: $z_2 \in \mathbb{R}^h$
- Output: $z_3 = W^{(2)}z_2 + b^{(2)}$
  - $W^{(2)} \in \mathbb{R}^{K \times h}$
  - $b^{(2)} \in \mathbb{R}^K$
  - $z_3 \in \mathbb{R}^K$

Activation g ($\sigma(\cdot)$):

- Input: $z_3 \in \mathbb{R}^K$
- Output: $\hat{y} = g(z_3) = \sigma(z_3)$
  - $\hat{y} \in \mathbb{R}^K$
  - $\hat{y}$ represents the predicted probabilities for each class.

Changes in Equations for Forward Pass (Part 1.2 (c)):

The backward pass calculations need to be updated to reflect the new activation functions $f = \tanh$ and $g = \sigma$. The loss function is

$$\ell_{\text{MSE}}(\hat{y}, y) = \|\hat{y} - y\|^2.$$

Chain structure. The chain of derivatives remains the same:

$$\frac{\partial \ell}{\partial z_3} \rightarrow \frac{\partial \ell}{\partial z_2} \rightarrow \frac{\partial \ell}{\partial z_1}.$$

The chain of derivatives expands as follows:

$$\boxed{\frac{\partial \ell}{\partial \mathbf{z_3}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}}$$

$$\boxed{\frac{\partial \ell}{\partial \mathbf{z_2}} = (\mathbf{W^{(2)}})^\top \frac{\partial \ell}{\partial \mathbf{z_3}}}$$

$$\boxed{\frac{\partial \ell}{\partial \mathbf{z_1}} = \left((\mathbf{W^{(2)}})^\top \frac{\partial \ell}{\partial \mathbf{z_3}}\right) \frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}}$$

Parameter-gradient forms. The gradients with respect to the parameters become:

$$\boxed{\frac{\partial \ell}{\partial \mathbf{b^{(1)}}} = \left((\mathbf{W^{(2)}})^\top \left(\frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}\right)\right) \left(\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}\right)}$$

$$\boxed{\frac{\partial \ell}{\partial \mathbf{W^{(1)}}} = \left(\left((\mathbf{W^{(2)}})^\top \left(\frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}\right)\right) \left(\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}\right)\right) \mathbf{x}^\top}$$

$$\boxed{\frac{\partial \ell}{\partial \mathbf{b^{(2)}}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}}$$

$$\boxed{\frac{\partial \ell}{\partial \mathbf{W^{(2)}}} = \left(\frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}\right) \mathbf{z}_2^\top}$$

Local Jacobians. Only the local Jacobians inside the chain change (listed in part (d) below).

Changes in Elements (Part 1.2 (d))

Elements of $\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}$:

- This is the Jacobian of the hidden layer's activation function $f(z_1) = \tanh(z_1)$.
- Since the tanh function is applied element-wise to z_1, the Jacobian is a diagonal matrix.
- The derivative of $\tanh(x)$ is $1 - \tanh^2(x)$.
- The element at the i-th row and j-th column is:

$$\boxed{\left[\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}}\right]_{ij} = \begin{cases} 1 - \tanh^2(z_{1,i}) & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}}$$

- Thus, the Jacobian is diagonal:

$$\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}} = \begin{bmatrix} 1 - \tanh^2((z_1)_1) & 0 & \cdots & 0 \\ 0 & 1 - \tanh^2((z_1)_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 - \tanh^2((z_1)_m) \end{bmatrix}.$$

- This Jacobian is of size $m \times m$.

Elements of $\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}}$:

- This is the Jacobian of the output layer's activation function $g(z_3) = \sigma(z_3)$ (sigmoid).
- Since the sigmoid function is applied element-wise to z_3, the Jacobian is a diagonal matrix.

Derivation of the derivative of $\sigma(u)$

The logistic sigmoid function is defined as:

$$\sigma(u) = \frac{1}{1+e^{-u}}.$$

To find its derivative $\frac{d}{du}\sigma(u)$, we apply the quotient rule:

$$\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2},$$

where $f(u) = 1$, $g(u) = 1 + e^{-u}$, $f'(u) = 0$, and $g'(u) = -e^{-u}$.
Applying the quotient rule:

$$\frac{d}{du}\sigma(u) = \frac{(0)(1+e^{-u}) - (1)(-e^{-u})}{(1+e^{-u})^2} = \frac{e^{-u}}{(1+e^{-u})^2}.$$

Now, since

$$\sigma(u) = \frac{1}{1+e^{-u}},$$

we can rewrite the derivative as:

$$\frac{e^{-u}}{(1+e^{-u})^2} = \sigma(u) \cdot \frac{e^{-u}}{1+e^{-u}}.$$

Noting that

$$\frac{e^{-u}}{1+e^{-u}} = 1 - \sigma(u),$$

we obtain the familiar form:

$$\frac{d}{du}\sigma(u) = \sigma(u)(1-\sigma(u)).$$

- Therefore, the derivative of $\sigma(x)$ is $\sigma(x)(1-\sigma(x))$.
- The Jacobian in expanded form is:

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial z_{3,1}} & \cdots & \frac{\partial \hat{y}_1}{\partial z_{3,K}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_K}{\partial z_{3,1}} & \cdots & \frac{\partial \hat{y}_K}{\partial z_{3,K}} \end{bmatrix}.$$

- Since $\hat{y}_i = \sigma(z_{3,i})$, we have

$$\frac{\partial \hat{y}_i}{\partial z_{3,j}} = \begin{cases} \hat{y}_i(1-\hat{y}_i), & i = j, \\ 0, & i \neq j, \end{cases}$$

so the Jacobian simplifies to:

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} = \begin{bmatrix} \hat{y}_1(1-\hat{y}_1) & 0 & \cdots & 0 \\ 0 & \hat{y}_2(1-\hat{y}_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{y}_K(1-\hat{y}_K) \end{bmatrix}.$$

- This Jacobian can be concisely represented as:

$$\boxed{\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} = \operatorname{diag}\big(\hat{y} \odot (1-\hat{y})\big) \in \mathbb{R}^{K \times K}.}$$

Elements of $\frac{\partial \ell}{\partial \hat{\mathbf{y}}}$ (using Mean Squared Error loss):

- The Mean Squared Error (MSE) loss function: $\ell = \sum_{i=1}^{K} (\hat{y}_i - y_i)^2$.
- Unchanged Gradient: The loss derivative w.r.t. $\hat{y}$ is not affected:

$$\boxed{\frac{\partial \ell}{\partial \hat{\mathbf{y}}} = 2(\hat{y} - y)}$$

-

1.3 (b): Changes for Classification with Binary Cross-Entropy (BCE) Loss

If we switch to Binary Cross-Entropy (BCE) loss:

$$\ell_{\text{BCE}}(\hat{y}, y) = \frac{1}{K} \sum_{i=1}^{K} -[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

The activation functions $f = \tanh$ and $g = \sigma$ are kept as described in Part 1.3 (a).

Changes in Equations for Forward Pass (Part 1.2 (b))

The forward pass equations remain exactly the same as in Part 1.3 (a) because the activation functions are unchanged.

Changes in Equations for Backward Pass (Part 1.2 (c))

The major change is in the gradient of the loss with respect to the output $\hat{y}$. The chain rule structure and parameter-gradient forms remain unchanged from Part 1.3(a), but the expression for $\frac{\partial \ell}{\partial \hat{y}}$ differs.

Gradient w.r.t. $\hat{y}$ (changed): For BCE loss,

$$\frac{\partial \ell_{\text{BCE}}}{\partial \hat{\mathbf{y}}_\mathbf{i}} = \frac{1}{K} \left(-\frac{y_i}{\hat{y}_i} + \frac{1 - y_i}{1 - \hat{y}_i} \right),$$

and in vector form,

$$\frac{\partial \ell_{\text{BCE}}}{\partial \hat{\mathbf{y}}} = \frac{1}{K} \left(-\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} \right),$$

where divisions are element-wise.

Downstream Gradients (unchanged chain structure):

$$\frac{\partial \ell}{\partial \mathbf{z_3}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \circ \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} = \left(\frac{1}{K} \left(-\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} \right) \right) \circ (\hat{y} \circ (1 - \hat{y})).$$

Example — Gradients w.r.t. parameters:

$$\frac{\partial \ell}{\partial \mathbf{W^{(2)}}} = \left(\frac{\partial \ell}{\partial \mathbf{z_3}} \right) z_2^\top, \quad \frac{\partial \ell}{\partial \mathbf{b^{(2)}}} = \frac{\partial \ell}{\partial \mathbf{z_3}}.$$

—

Changes in Elements (Part 1.2 (d))

Unchanged:

$$\frac{\partial \mathbf{z_2}}{\partial \mathbf{z_1}} = \text{diag}(1 - \tanh^2(z_1)) \in \mathbb{R}^{h \times h}, \quad \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z_3}} = \text{diag}(\hat{y} \odot (1 - \hat{y})) \in \mathbb{R}^{K \times K}.$$

Changed:

$$\boxed{\frac{\partial \ell}{\partial \hat{\mathbf{y}}} = \frac{1}{K} \left(-\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} \right) \in \mathbb{R}^{K \times 1}}$$

1.3 (c): Choice of $f(\cdot) = (\cdot)^+$ with $g = \tanh$

We choose the hidden activation as ReLU and the output activation as tanh:

$$f(z) = \max(0, z), \quad f'(z) = \begin{cases} 1, & z > 0, \\ 0, & z \leq 0, \end{cases}$$

$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad g'(z) = 1 - \tanh^2(z).$$

Explanation of Benefit: Using $f(\cdot) = (\cdot)^+$ (ReLU) for the hidden layer and $g = \tanh$ for the output layer combines the strengths of both activations:

1. **Mitigating Vanishing Gradients:** The derivative of $\tanh(u)$ is $1-\tanh^2(u)$, which approaches 0 as $|u| \to \infty$, causing vanishing gradients in deep networks. ReLU, on the other hand, has derivative $f'(z) = 1$ for $z > 0$, which maintains strong gradient flow and prevents gradients from vanishing in hidden layers.
2. **Sparsity and Efficiency:** ReLU outputs 0 for $z \leq 0$, leading to sparse activations. This improves computational efficiency and enhances feature selectivity.
3. **Avoiding Saturation in the Hidden Layer:** Unlike tanh, which saturates for large $|z|$, ReLU avoids saturation on the positive side, allowing more effective gradient propagation during back-propagation.
4. **Output Layer Characteristics:** Keeping tanh at the output ensures bounded predictions:

   $$\hat{y} = g(z_3) = \tanh(z_3) \in [-1, 1],$$

   which is desirable in tasks where normalized or bounded outputs are required (e.g., regression to a fixed range, binary classification with targets in $[-1, 1]$).

Summary: ReLU in hidden layers improves trainability of deep networks by avoiding vanishing gradients and introducing sparsity, while tanh in the output layer ensures bounded, interpretable outputs. This hybrid choice balances optimization stability with meaningful output constraints.

1.4 Deriving Loss Functions

We derive loss functions whose gradient (or subgradient) with respect to each weight w_i yields the common update

$$w_i \leftarrow w_i + \eta\,(y - \hat{y})\,x_i,$$

i.e. a gradient-descent style relation

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i} \quad \Longrightarrow \quad \frac{\partial L}{\partial w_i} = -(y - \hat{y})\,x_i.$$

Define

$$z := b + \sum_{j=1}^{d} w_j x_j, \qquad \hat{y} = f(z).$$

By the chain rule

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial L}{\partial z}\,x_i,$$

so the desired update implies

$$\frac{\partial L}{\partial z} = \hat{y} - y.$$

Thus for differentiable f we have the ordinary differential equation in z

$$\frac{dL}{dz} = f(z) - y,$$

and integrating gives

$$L(z) = \int f(z)\,dz - yz + C.$$

Below we apply this (and treat the perceptron case via a standard convex surrogate) to the three requested models.

(1) Perceptron: $\hat{y} = \text{sign}(z)$. The sign function is not differentiable and the update that appears in the perceptron algorithm

$$w_i \leftarrow w_i + \eta\, y\, x_i \quad \text{(only when } y\hat{y} \le 0\text{)}$$

is obtained from the perceptron (hinge-like) surrogate loss. A commonly used convex surrogate is the perceptron loss

$$L_{\text{perc}}(z, y) \;=\; \max(0,\, -yz).$$

Compute its subgradient w.r.t. z:

$$\partial_z L_{\text{perc}}(z, y) = \begin{cases} 0, & yz > 0, \\ -\,y, & yz < 0, \\ \text{any value in } [-y, 0], & yz = 0. \end{cases}$$

Then the subgradient w.r.t. w_i is

$$\partial_{w_i} L_{\text{perc}}(z, y) = \partial_z L_{\text{perc}}(z, y)\, x_i = \begin{cases} 0, & yz > 0, \\ -\,yx_i, & yz < 0. \end{cases}$$

A subgradient-descent update $w_i \leftarrow w_i - \eta\, \partial_{w_i} L_{\text{perc}}$ yields

$$w_i \leftarrow \begin{cases} w_i, & yz > 0, \\ w_i + \eta\, yx_i, & yz < 0, \end{cases}$$

which reproduces the perceptron weight update (changes occur only for misclassified or margin-failure examples). Hence the perceptron loss that yields the stated update is

$$\boxed{L_{\text{perc}}(z, y) = \max(0,\, -yz)}.$$

(2) Adaline / Least Mean Squares (LMS): $\hat{y} = z$. Here $f(z) = z$. Using the integral formula

$$L(z) = \int f(z)\, dz - yz + C = \int z\, dz - yz + C = \tfrac{1}{2}z^2 - yz + C.$$

Up to an additive constant this is the familiar squared error. Rewriting,

$$L(z, y) = \tfrac{1}{2}z^2 - yz = \tfrac{1}{2}(z^2 - 2yz) = \tfrac{1}{2}(z - y)^2 - \tfrac{1}{2}y^2.$$

Dropping the constant $-\frac{1}{2}y^2$ we obtain the standard form

$$\boxed{L_{\text{LMS}}(z, y) = \tfrac{1}{2}(y - z)^2}.$$

(3) "Logistic" with $\hat{y} = \tanh(z)$. Here $f(z) = \tanh(z)$. Use the indefinite integral

$$\int \tanh(z)\, dz = \ln\cosh(z) + C,$$

because $\dfrac{d}{dz} \ln\cosh z = \tanh z$. Applying the general formula:

$$L(z) = \int \tanh(z)\, dz - yz + C = \ln\cosh(z) - yz + C.$$

Dropping the additive constant,

$$\boxed{L_{\tanh}(z, y) = \ln\cosh(z) - yz}.$$

Short summary (per training example): As $z = b + \sum_{j=1}^{d} w_j x_j$. Then

Perceptron:

$$L_{\text{perc}}\Big(b + \sum_j w_j x_j,\ y\Big) = \max\Big(0,\ -y\,(b + \sum_j w_j x_j)\Big),$$
$$\hat{y} = \text{sign}\Big(b + \sum_j w_j x_j\Big),$$
$$w_i \leftarrow w_i + \eta\,(y - \hat{y})\,x_i, \qquad b \leftarrow b + \eta\,(y - \hat{y}),$$

Adaline / LMS:

$$L_{\text{LMS}}\Big(b + \sum_j w_j x_j,\ y\Big) = \tfrac{1}{2}\Big(y - (b + \sum_j w_j x_j)\Big)^2,$$
$$\hat{y} = b + \sum_j w_j x_j,$$
$$w_i \leftarrow w_i + \eta\,(y - \hat{y})\,x_i, \qquad b \leftarrow b + \eta\,(y - \hat{y}),$$

Logistic (tanh link):

$$L_{\tanh}\Big(b + \sum_j w_j x_j,\ y\Big) = \ln\cosh\big(b + \sum_j w_j x_j\big) \ -\ y\,(b + \sum_j w_j x_j),$$
$$\hat{y} = \tanh\Big(b + \sum_j w_j x_j\Big),$$
$$w_i \leftarrow w_i + \eta\,(y - \hat{y})\,x_i, \qquad b \leftarrow b + \eta\,(y - \hat{y}).$$

Tabular summary (per training example): Let $z = b + \sum_{j=1}^{d} w_j x_j$. The prediction $\hat{y}$, loss $L(z, y)$, and parameter updates are:

Model	**Prediction $\hat{y}$**	**Loss $L(z, y)$**	**Update rules**
Perceptron	$\hat{y} = \text{sign}(z)$	$L_{\text{perc}}(z, y) = \max(0,\ -yz)$	$w_i \leftarrow w_i + \eta\,(y - \hat{y})\,x_i$, $b \leftarrow b + \eta\,(y - \hat{y})$
Adaline / LMS	$\hat{y} = z$	$L_{\text{LMS}}(z, y) = \frac{1}{2}(y - z)^2$	$w_i \leftarrow w_i + \eta\,(y - \hat{y})\,x_i$, $b \leftarrow b + \eta\,(y - \hat{y})$
Logistic (tanh link)	$\hat{y} = \tanh(z)$	$L_{\tanh}(z, y) = \ln\cosh(z) - yz$	$w_i \leftarrow w_i + \eta\,(y - \hat{y})\,x_i$, $b \leftarrow b + \eta\,(y - \hat{y})$

1.5 Conceptual Questions

(a) Why is Softmax actually Softargmax?

Argmax: For a score vector $z \in \mathbb{R}^K$, the argmax operator produces a one-hot vector:

$$\text{argmax}(z)_i = \begin{cases} 1, & i = \arg\max_j z_j, \\ 0, & \text{otherwise.} \end{cases}$$

Softmax: The softmax function is defined as

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad i = 1, \ldots, K,$$

which produces a probability distribution over all entries, with larger logits z_i corresponding to higher probabilities.

Why "Softargmax"? The softmax is a continuous, differentiable relaxation of the discrete arg max operator:

$$\lim_{\beta \to \infty} \text{softmax}(\beta z) = \arg\max(z).$$

Here β is a scaling parameter. More generally, with temperature scaling

$$\text{softmax}_\tau(z)_i = \frac{e^{z_i/\tau}}{\sum_j e^{z_j/\tau}}, \quad \tau > 0,$$

we see that:

- As $\tau \to 0$, the distribution sharpens and converges to a one-hot vector given by $\arg\max(z)$.
- As $\tau \to \infty$, the distribution becomes uniform across all indices.

Thus, the softmax does not pick a single maximum entry, but instead produces "soft" weights across all entries, with the maximum receiving the highest probability. In this sense, it is a *softened version of argmax*, hence the name **softargmax**.

(b) (3 pt) Draw the computational graph for

$$\begin{aligned} a &= x * y + z, \\ b &= (x + x) * a, \\ w &= a * b, \end{aligned}$$

with inputs $x, y, z \in \mathbb{R}$ and output $w \in \mathbb{R}$. Use symbols x, y, z, a, b, w and operators $*, +$.

Computational Graph

We want to represent the following computations:

$$a = x * y + z, \quad b = (x + x) * a, \quad w = a * b$$

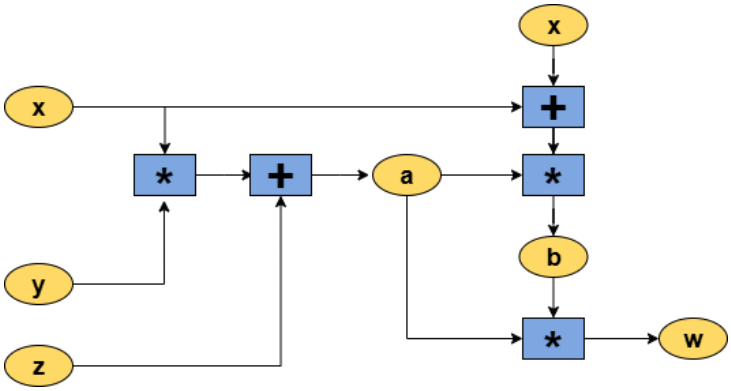

Figure 1: *
Computational graph for $w = a * b$ with intermediate nodes a and b.

(c) (2 pt) Draw the graph of the *derivative* for each function and give the analytic form.

Derivatives of Activation Functions

ReLU

$$\text{ReLU}(x) = \max(0, x), \qquad \frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1, & x > 0, \\ 0, & x < 0, \end{cases} \quad \text{(undefined at } x = 0\text{, subgradient} \in [0, 1]).$$

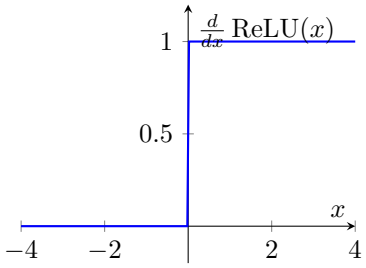

Figure 2: *
Derivative of ReLU

—

LeakyReLU ($\alpha = 0.01$)

$$\text{LeakyReLU}(x) = \begin{cases} x, & x > 0, \\ 0.01x, & x \leq 0, \end{cases} \qquad \frac{d}{dx}\text{LeakyReLU}(x) = \begin{cases} 1, & x > 0, \\ 0.01, & x < 0. \end{cases}$$

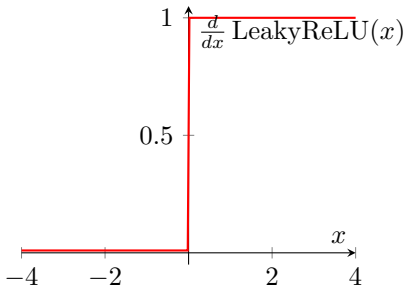

Figure 3: *
Derivative of LeakyReLU with $\alpha = 0.01$

—

Softplus

$$\text{Softplus}(x) = \ln(1 + e^x), \qquad \frac{d}{dx}\text{Softplus}(x) = \frac{1}{1 + e^{-x}}.$$

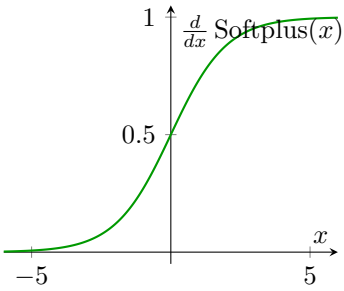

Figure 4: *
Derivative of Softplus (Sigmoid)

—

GELU (tanh-based approximation)

$$\mathrm{GELU}(x) \approx \tfrac{1}{2}x\Big(1+\tanh\big(\sqrt{\tfrac{2}{\pi}}(x+0.044715x^3)\big)\Big),$$

$$\frac{d}{dx}\,\mathrm{GELU}(x) \approx \tfrac{1}{2}\Big(1+\tanh(u(x))\Big) + \tfrac{1}{2}x\Big(1-\tanh^2(u(x))\Big)\,u'(x),$$

where

$$u(x) = \sqrt{\tfrac{2}{\pi}}(x+0.044715x^3), \qquad u'(x) = \sqrt{\tfrac{2}{\pi}}\big(1+3\cdot 0.044715\,x^2\big).$$

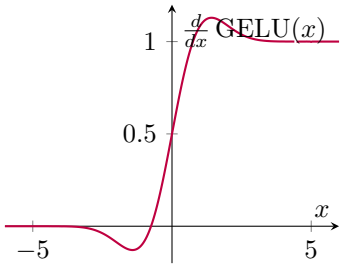

Figure 5: *
Derivative of GELU (tanh-based approximation)

(d) (3pt) Four different types of linear transformations and their role

A linear transformation $T : \mathbb{R}^n \to \mathbb{R}^m$ can always be written as $T(x) = Ax$ for some matrix $A \in \mathbb{R}^{m\times n}$. Four common types (with short characterization and matrix form) are:

- **Scaling:** Scaling changes the magnitude of a vector but not its direction (except when scaled by a negative constant, which also flips the direction).
  Transformation: $T(x) = \alpha x,\ \alpha \in \mathbb{R}$. If $\alpha > 1$, the vector is stretched; if $0 < \alpha < 1$, the vector is compressed; and if $\alpha < 0$, the vector is both scaled and reflected.
  Example: Multiplying a 2D vector (x, y) by $\alpha = 2$ doubles its length, while multiplying by $\alpha = 0.5$ halves its length.

- **Rotation:** orthogonal matrix with $A^\top A = I$ and $\det(A) = 1$. Effect: rotates vectors without changing their length. Example: In $\mathbb{R}^2$, a counter-clockwise rotation by angle θ is

  $$R_\theta = \begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}.$$

  Applying $T(x) = R_\theta x$ rotates every point in the plane by θ around the origin. For instance, rotating $(1, 0)$ by $\theta = \pi/2$ transforms it into $(0, 1)$.

- **Reflection:** orthogonal matrix with $\det(A) = -1$. Effect: mirrors vectors across a subspace. Example: Reflecting across the y-axis in $\mathbb{R}^2$ uses

  $$T(x) = \begin{bmatrix}-1 & 0 \\ 0 & 1\end{bmatrix} x,$$

  which negates the x-coordinate while leaving the y-coordinate unchanged. For example, $(3, 2)$ becomes $(-3, 2)$.

- **Shear (and Projection):**
  • Shear: e.g. $A = \begin{bmatrix}1 & k \\ 0 & 1\end{bmatrix}$ (in 2D) slants one axis relative to another. Example: If $k = 1$, the unit square with vertices $(0, 0), (1, 0), (0, 1), (1, 1)$ becomes a parallelogram.
  • Projection: idempotent matrix P with $P^2 = P$ (projects onto a subspace). Example: Projection onto the x-axis maps $(3, 2)$ to $(3, 0)$.

Role in neural networks.

- *Linear (affine) layers.* A neural-network linear layer computes an affine map

$$\text{Linear}(x) = Wx + b,$$

  which is a composition of a linear transformation $x \mapsto Wx$ and a translation by b. Linear layers change basis, rescale, rotate, project, or mix input coordinates and are essential to reshape the representation at each layer.

- *Nonlinear activations.* Element-wise nonlinearities (ReLU, sigmoid, tanh, etc.) inject nonlinearity between affine maps. Without them, a stack of affine layers collapses to a single affine map:

$$W^{(2)}(W^{(1)}x + b^{(1)}) + b^{(2)} = (W^{(2)}W^{(1)})x + (W^{(2)}b^{(1)} + b^{(2)}),$$

  so multiple layers give no extra modelling power. Nonlinearities allow the network to approximate complex, non-linear functions (this is the reason deep networks can learn hierarchical, compositional features).

(e) (3pt) Mathematical definition of training with MSE loss

Let the network be $F_\theta : \mathbb{R}^n \to \mathbb{R}^K$ parameterized by θ. Given dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ with targets $y_i \in \mathbb{R}^K$, the empirical MSE (mean squared error) objective is

$$\mathcal{L}(\theta) \;=\; \frac{1}{N}\sum_{i=1}^{N} \ell_{\text{MSE}}\big(F_\theta(x_i),\, y_i\big) \;=\; \frac{1}{N}\sum_{i=1}^{N} \big\|F_\theta(x_i) - y_i\big\|_2^2.$$

Training the network is the optimization problem

$$\theta^\star \;=\; \arg\min_\theta \; \mathcal{L}(\theta).$$

A standard iterative method (gradient descent / SGD) uses updates such as

$$\theta \leftarrow \theta - \eta\, \nabla_\theta \mathcal{L}(\theta),$$

or (mini-batch) stochastic gradient update for a batch $\mathcal{B}$:

$$\theta \leftarrow \theta - \eta\, \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \nabla_\theta \big\|F_\theta(x_i) - y_i\big\|_2^2,$$

where $\eta > 0$ is the learning rate.