CSCI-GA 2572: Deep Learning, Fall 2025

Backpropagation

Aditeya Baral
N19186654

Problem 1.2: Regression Task

Problem a

The five canonical steps to train a model with SGD on a single batch of data are,

1. **Forward pass.** Compute predictions $\hat{y}$ for the batch inputs x:

$$\hat{y} = \text{model}(x; \theta)$$

2. **Compute loss.** Evaluate the loss function comparing predictions $\hat{y}$ with targets y:

$$\ell(\theta; x, y) = \mathcal{L}(\hat{y}, y) = \|\hat{y} - y\|^2$$

3. **Zero gradients.** Reset/Clear previously accumulated gradients:

$$\nabla_\theta \leftarrow 0$$

4. **Backward pass.** Compute gradients of the loss with respect to the parameters:

$$\nabla_\theta \ell = \frac{\partial \ell}{\partial \theta}$$

5. **Parameter update with SGD.** Update the parameters with learning rate η:

$$\theta \leftarrow \theta - \eta \nabla_\theta \ell$$

These steps are repeated across batches over multiple epochs until convergence.

Problem b

The forward pass proceeds layer by layer as follows,

1. **Layer 1 (Linear):** *Input $x \in \mathbb{R}^n$, weights and bias*

$$W^{(1)} \in \mathbb{R}^{h \times n}, \qquad b^{(1)} \in \mathbb{R}^h,$$

where h is the number of hidden units in the first layer, *and output*

$$z_1 = W^{(1)} x + b^{(1)} \in \mathbb{R}^h.$$

2. **Layer 2 (Nonlinearity f):** *Input $z_1 \in \mathbb{R}^h$, apply element-wise*

$$f(z_1) = 5\,\mathrm{ReLU}(z_1),$$

producing

$$z_2 = f(z_1) = 5\,\mathrm{ReLU}(z_1) \in \mathbb{R}^h.$$

3. **Layer 3 (Linear):** *Input $z_2 \in \mathbb{R}^h$, weights and bias*

$$W^{(2)} \in \mathbb{R}^{K\times h}, \qquad b^{(2)} \in \mathbb{R}^K,$$

and output

$$z_3 = W^{(2)} z_2 + b^{(2)} \in \mathbb{R}^K.$$

4. **Layer 4 (Nonlinearity g):** *Input $z_3 \in \mathbb{R}^K$, apply element-wise*

$$g(z_3) = z_3,$$

producing

$$\hat{y} = g(z_3) = z_3 \in \mathbb{R}^K.$$

Problem c

Backward-pass gradients comprise,

1. **Output layer:**

$$\frac{\partial \ell}{\partial z_3} \;=\; \frac{\partial \ell}{\partial \hat{y}}\,\frac{\partial \hat{y}}{\partial z_3} \;=\; 2(\hat{y}-y)\,\frac{\partial \hat{y}}{\partial z_3} \;=\; 2(\hat{y}-y)$$

here g is the identity, so $\frac{\partial \hat{y}}{\partial z_3} = I_K$, hence $\frac{\partial \ell}{\partial z_3} = 2(\hat{y}-y)$

2. **Linear$_2$:**

$$\frac{\partial \ell}{\partial W^{(2)}} \;=\; \Big(\frac{\partial \ell}{\partial z_3}\Big)\, z_2^\top, \qquad \frac{\partial \ell}{\partial b^{(2)}} \;=\; \frac{\partial \ell}{\partial z_3}$$

Backprop into the previous activation (output of f):

$$\frac{\partial \ell}{\partial z_2} \;=\; \big(W^{(2)}\big)^\top\,\frac{\partial \ell}{\partial z_3}$$

3. **Gradients of nonlinearity f :**

$$\frac{\partial \ell}{\partial z_1} \;=\; \frac{\partial z_2}{\partial z_1}\,\frac{\partial \ell}{\partial z_2} \;=\; \big(5\cdot \mathbf{1}_{\{z_1>0\}}\big)\odot \frac{\partial \ell}{\partial z_2}$$

for $f(z_1) = 5\,\mathrm{ReLU}(z_1)$, $\frac{\partial z_2}{\partial z_1} = \mathrm{diag}(5\cdot \mathbf{1}_{\{z_1>0\}})$, so $\frac{\partial \ell}{\partial z_1} = \big(5\cdot \mathbf{1}_{\{z_1>0\}}\big)\odot \frac{\partial \ell}{\partial z_2}$

4. **Linear$_1$:**

$$\frac{\partial \ell}{\partial W^{(1)}} \;=\; \Big(\frac{\partial \ell}{\partial z_1}\Big)\, x^\top, \qquad \frac{\partial \ell}{\partial b^{(1)}} \;=\; \frac{\partial \ell}{\partial z_1}, \qquad \frac{\partial \ell}{\partial x} \;=\; \big(W^{(1)}\big)^\top\,\frac{\partial \ell}{\partial z_1}$$

Problem d

1. **Derivative of the nonlinearity f:**

$$\frac{\partial z_2}{\partial z_1} = \begin{bmatrix} 5 \cdot \mathbf{1}_{\{z_{1,1}>0\}} & 0 & \cdots & 0 \\ 0 & 5 \cdot \mathbf{1}_{\{z_{1,2}>0\}} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 5 \cdot \mathbf{1}_{\{z_{1,m}>0\}} \end{bmatrix} = \operatorname{diag}\left(5 \cdot \mathbf{1}_{\{z_1>0\}}\right) \in \mathbb{R}^{m \times m}.$$

   where $\mathbf{1}_{\{z_1>0\}} = [\mathbf{1}_{\{z_{1,1}>0\}}, \ldots, \mathbf{1}_{\{z_{1,m}>0\}}]^\top$, $W^{(1)} \in \mathbb{R}^{m \times n}$, and m = number of hidden units in the first linear layer.

2. **Derivative of the final activation g:**

$$\frac{\partial \hat{y}}{\partial z_3} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} = I_K \in \mathbb{R}^{K \times K}.$$

3. **Gradient of the loss with respect to the output:**

$$\frac{\partial \ell}{\partial \hat{y}} = \begin{bmatrix} 2(\hat{y}_1 - y_1) \\ 2(\hat{y}_2 - y_2) \\ \vdots \\ 2(\hat{y}_K - y_K) \end{bmatrix} \in \mathbb{R}^K.$$

Problem 1.3: Classification Task

Problem a

For the classification task with $f = \tanh$ and $g = \sigma$ (sigmoid), using MSE loss, the changes are,

1. **Forward pass:**

   Changed equations:

   $$z_2 = f(z_1) = \tanh(z_1), \qquad \hat{y} = g(z_3) = \sigma(z_3).$$

   All other forward equations remain the same:

   $$z_1 = W^{(1)}x + b^{(1)}, \qquad z_3 = W^{(2)}z_2 + b^{(2)}.$$

2. **Backward pass:**

   Changed equations:

   $$\frac{\partial z_2}{\partial z_1} = 1 - \tanh^2(z_1), \qquad \frac{\partial \hat{y}}{\partial z_3} = \sigma(z_3)(1 - \sigma(z_3)).$$

   All other backward equations remain the same:

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

   $$\frac{\partial \ell}{\partial z_3} = \frac{\partial \hat{y}}{\partial z_3} \odot \frac{\partial \ell}{\partial \hat{y}}, \quad \frac{\partial \ell}{\partial W^{(2)}} = \frac{\partial \ell}{\partial z_3} z_2^\top, \quad \frac{\partial \ell}{\partial b^{(2)}} = \frac{\partial \ell}{\partial z_3},$$

   $$\frac{\partial \ell}{\partial z_2} = (W^{(2)})^\top \frac{\partial \ell}{\partial z_3}, \quad \frac{\partial \ell}{\partial z_1} = \frac{\partial z_2}{\partial z_1} \odot \frac{\partial \ell}{\partial z_2},$$

   $$\frac{\partial \ell}{\partial W^{(1)}} = \frac{\partial \ell}{\partial z_1} x^\top, \quad \frac{\partial \ell}{\partial b^{(1)}} = \frac{\partial \ell}{\partial z_1}, \quad \frac{\partial \ell}{\partial x} = (W^{(1)})^\top \frac{\partial \ell}{\partial z_1}.$$

3. **Derivatives:**

   Changed equations:

   $$\frac{\partial z_2}{\partial z_1} = \begin{bmatrix} 1 - \tanh^2(z_{1,1}) & 0 & \cdots & 0 \\ 0 & 1 - \tanh^2(z_{1,2}) & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 - \tanh^2(z_{1,m}) \end{bmatrix} = \operatorname{diag}\big(1 - \tanh^2(z_1)\big) \in \mathbb{R}^{m \times m},$$

   $$\frac{\partial \hat{y}}{\partial z_3} = \begin{bmatrix} \sigma(z_{3,1})(1 - \sigma(z_{3,1})) & 0 & \cdots & 0 \\ 0 & \sigma(z_{3,2})(1 - \sigma(z_{3,2})) & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma(z_{3,K})(1 - \sigma(z_{3,K})) \end{bmatrix} = \operatorname{diag}\big(\sigma(z_3)(1 - \sigma(z_3))\big) \in \mathbb{R}^{K \times K}.$$

   All other gradient elements remain the same:

   $$\frac{\partial \ell}{\partial \hat{y}} = 2(\hat{y} - y) = \begin{bmatrix} 2(\hat{y}_1 - y_1) \\ 2(\hat{y}_2 - y_2) \\ \vdots \\ 2(\hat{y}_K - y_K) \end{bmatrix} \in \mathbb{R}^K.$$

Problem b

Since we now use the BCE loss, the changes made are,

1. **Forward pass:**

   No change to forward equations beyond those in 1.3(a):

   $$z_1 = W^{(1)}x + b^{(1)}, \qquad z_2 = \tanh(z_1), \qquad z_3 = W^{(2)}z_2 + b^{(2)}, \qquad \hat{y} = \sigma(z_3).$$

2. **Backward pass:**

   Changed equations:

   $$\frac{\partial \ell_{BCE}}{\partial \hat{y}} = \frac{1}{K}\frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \qquad \in \mathbb{R}^K,$$

   $$\frac{\partial \ell_{BCE}}{\partial z_3} = \frac{\partial \ell_{BCE}}{\partial \hat{y}} \odot \frac{\partial \hat{y}}{\partial z_3} = \frac{1}{K}\frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \odot \big(\hat{y}(1-\hat{y})\big) = \frac{1}{K}(\hat{y}-y),$$

   All other backward equations stay the same as 1.3(a):

   $$\frac{\partial \ell_{BCE}}{\partial W^{(2)}} = \frac{\partial \ell_{BCE}}{\partial z_3} z_2^\top, \qquad \frac{\partial \ell_{BCE}}{\partial b^{(2)}} = \frac{\partial \ell_{BCE}}{\partial z_3},$$

   $$\frac{\partial \ell_{BCE}}{\partial z_2} = (W^{(2)})^\top \frac{\partial \ell_{BCE}}{\partial z_3}, \qquad \frac{\partial \ell_{BCE}}{\partial z_1} = \big(1-\tanh^2(z_1)\big) \odot \frac{\partial \ell_{BCE}}{\partial z_2},$$

   $$\frac{\partial \ell_{BCE}}{\partial W^{(1)}} = \frac{\partial \ell_{BCE}}{\partial z_1} x^\top, \qquad \frac{\partial \ell_{BCE}}{\partial b^{(1)}} = \frac{\partial \ell_{BCE}}{\partial z_1}, \qquad \frac{\partial \ell_{BCE}}{\partial x} = (W^{(1)})^\top \frac{\partial \ell_{BCE}}{\partial z_1}.$$

3. **Derivatives:**

   Changed equations:

   $$\frac{\partial \ell_{BCE}}{\partial \hat{y}} = \frac{1}{K}\frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \in \mathbb{R}^K,$$

   $$\frac{\partial \ell_{BCE}}{\partial z_3} = \frac{1}{K}(\hat{y}-y) \in \mathbb{R}^K.$$

   All other gradient elements stay the same as 1.3(a):

   $$\frac{\partial z_2}{\partial z_1} = \begin{bmatrix} 1-\tanh^2(z_{1,1}) & 0 & \cdots & 0 \\ 0 & 1-\tanh^2(z_{1,2}) & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1-\tanh^2(z_{1,m}) \end{bmatrix} \in \mathbb{R}^{m\times m},$$

   $$\frac{\partial \hat{y}}{\partial z_3} = \begin{bmatrix} \hat{y}_1(1-\hat{y}_1) & 0 & \cdots & 0 \\ 0 & \hat{y}_2(1-\hat{y}_2) & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{y}_K(1-\hat{y}_K) \end{bmatrix} \in \mathbb{R}^{K\times K}.$$

Problem c

Using $f(z_1) = \text{ReLU}(z_1) = (z_1)^+$ for the nonlinearity f is beneficial because,

1. **Avoids vanishing gradients:** Unlike tanh or sigmoid, ReLU does not saturate for positive inputs, so the derivative remains 1. Since it also does not have upper or lower bounds for positive inputs, hidden units are less likely to saturate, maintaining stronger gradient signals during training. This allows gradients to flow more effectively through deeper networks.
2. **Improves training of deeper networks:** Non-saturating activations help prevent the vanishing gradient problem, making it easier to train multiple layers.
3. **Introduces sparsity:** ReLU outputs zero for negative inputs, leading to sparse activations which can improve computational efficiency and sometimes generalization. Sparse, non-saturating activations encourage diverse feature representations in hidden layers, which can improve learning and generalization for complex tasks.
4. **Output layer remains** tanh**:** This ensures that the network output stays in the desired range (e.g., $[-1, 1]$) while benefiting from efficient gradient flow in hidden layers.
5. **Simpler derivative:** The derivative of ReLU is either 0 or 1, which simplifies the computation of gradients during backpropagation compared to tanh or sigmoid.

Problem 1.4: Deriving Loss Functions

Problem 1

Given the Perceptron model:

$$\hat{y} = \text{sign}\Big(b + \sum_{i=1}^{d} w_i x_i\Big), \qquad w_i \leftarrow w_i + \eta(y - \hat{y})x_i,$$

we derive the corresponding loss function as follows,

1. The Perceptron only updates its weights when the predicted label differs from the true label, i.e., when $y \neq \hat{y}$. Equivalently, this happens when the signed margin is non-positive:

$$y(b + w^\top x) \leq 0.$$

2. We define a Perceptron loss $\ell_{\text{perceptron}}$ such that

$$\frac{\partial \ell_{\text{perceptron}}}{\partial w_i} = \begin{cases} 0 & \text{if } y(b + w^\top x) > 0 \quad \text{(correct classification)} \\ -yx_i & \text{if } y(b + w^\top x) \leq 0 \quad \text{(misclassification)} \end{cases}.$$

3. A function whose gradient matches the above conditions is

$$\ell_{\text{perceptron}}(w) = \max\Big(0,\ -y(b + w^\top x)\Big).$$

   This loss is zero when the example is correctly classified, and grows linearly with the degree of misclassification otherwise. At the boundary case ($y(b + w^\top x) = 0$), the perceptron loss is non-differentiable and any subgradient between 0 and $-yx$ (for w), and between 0 and $-y$ (for b) is admissible.

4. The gradient of the Perceptron loss is thus

$$\frac{\partial \ell_{\text{perceptron}}}{\partial w} = \begin{cases} 0 & y(b + w^\top x) > 0 \\ -yx & y(b + w^\top x) \leq 0 \end{cases}.$$

   Applying gradient descent with learning rate η gives

$$w \leftarrow w - \eta \frac{\partial \ell_{\text{perceptron}}}{\partial w} = w + \eta yx \quad \text{(when misclassified)}.$$

   This is equivalent to the original Perceptron update rule $w \leftarrow w + \eta(y - \hat{y})x$ since when misclassified, $\hat{y} = -y$, making $(y - \hat{y}) = y - (-y) = 2y$. The factor of 2 difference doesn't affect the direction of the update, only its magnitude, and can be absorbed into the learning rate η making the update exactly equivalent. More precisely, if we use learning rate $\eta' = \eta/2$ in our derived loss function, we recover the exact original update rule, demonstrating that our loss captures the essential Perceptron behavior. Similarly, for the bias term, we have

$$\frac{\partial \ell_{\text{perceptron}}}{\partial b} = \begin{cases} 0 & y(b + w^\top x) > 0 \\ -y & y(b + w^\top x) \leq 0 \end{cases},$$

   so gradient descent gives $b \leftarrow b + \eta y$ when misclassified.

Final Perceptron loss:

$$\boxed{\ell_{\text{perceptron}}(w, b; x, y) = \max(0,\ -y(b + w^\top x))}.$$

Problem 2

Given the Adaline (Least Mean Squares) model:

$$\hat{y} = b + \sum_{i=1}^{d} w_i x_i = b + w^\top x, \qquad w_i \leftarrow w_i + \eta(y - \hat{y})x_i,$$

we derive the corresponding loss function as follows,

1. Adaline aims to minimize the difference between the predicted output $\hat{y}$ and the target y. The update occurs proportionally to the error $(y - \hat{y})$.

2. We define an Adaline loss ℓ_{adaline} such that

$$\frac{\partial \ell_{\text{adaline}}}{\partial w_i} = -(y - \hat{y})x_i, \quad \frac{\partial \ell_{\text{adaline}}}{\partial b} = -(y - \hat{y}),$$

   so that gradient descent with learning rate η reproduces the Adaline update:

$$w_i \leftarrow w_i - \eta \frac{\partial \ell_{\text{adaline}}}{\partial w_i} = w_i + \eta(y - \hat{y})x_i, \quad b \leftarrow b - \eta \frac{\partial \ell_{\text{adaline}}}{\partial b} = b + \eta(y - \hat{y})$$

3. A function whose gradient matches the above conditions is the squared error loss:

$$\ell_{\text{adaline}}(w, b; x, y) = \frac{1}{2}\big(y - (b + w^\top x)\big)^2,$$

   where the factor $\frac{1}{2}$ is conventional and simplifies the gradient expressions.

4. The gradient of the Adaline loss is thus

$$\frac{\partial \ell_{\text{adaline}}}{\partial w_i} = -(y - \hat{y})x_i, \quad \frac{\partial \ell_{\text{adaline}}}{\partial b} = -(y - \hat{y}),$$

   which exactly reproduces the Adaline update rule under gradient descent.

Final Adaline loss:

$$\boxed{\ell_{\text{adaline}}(w, b; x, y) = \frac{1}{2}\big(y - (b + w^\top x)\big)^2}$$

Problem 3

Given the Logistic Regression model with tanh activation:

$$\hat{y} = \tanh\left(b + \sum_{i=1}^{d} w_i x_i\right) = \tanh(b + w^\top x), \qquad w_i \leftarrow w_i + \eta(y - \hat{y})x_i,$$

we derive the corresponding loss function as follows,

1. Logistic Regression aims to minimize the discrepancy between the predicted output $\hat{y}$ and the target $y \in \{-1, +1\}$. The update occurs proportionally to the error $(y - \hat{y})$.

2. We can define a logistic loss ℓ_{logistic} such that

$$\frac{\partial \ell_{\text{logistic}}}{\partial w_i} = -(y - \hat{y})x_i, \quad \frac{\partial \ell_{\text{logistic}}}{\partial b} = -(y - \hat{y}),$$

   so that gradient descent with learning rate η reproduces the Logistic Regression update:

$$w_i \leftarrow w_i - \eta \frac{\partial \ell_{\text{logistic}}}{\partial w_i} = w_i + \eta(y - \hat{y})x_i, \quad b \leftarrow b - \eta \frac{\partial \ell_{\text{logistic}}}{\partial b} = b + \eta(y - \hat{y})$$

3. To find such a loss, we use the chain rule. Since $\hat{y} = \tanh(z)$ where $z = b + w^T x$:

$$\frac{\partial \ell_{\text{logistic}}}{\partial w_i} = \frac{\partial \ell_{\text{logistic}}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_i} = \frac{\partial \ell_{\text{logistic}}}{\partial \hat{y}} \cdot (1 - \hat{y}^2)x_i$$

   For this to equal $-(y - \hat{y})x_i$, we need:

$$\frac{\partial \ell_{\text{logistic}}}{\partial \hat{y}} = -\frac{y - \hat{y}}{1 - \hat{y}^2}$$

4. Integrating this expression:

$$\ell_{\text{logistic}}(\hat{y}) = \int -\frac{y - \hat{y}}{1 - \hat{y}^2} d\hat{y}$$

   This integral evaluates to:

$$\ell_{\text{logistic}}(\hat{y}) = -\frac{1}{2}[(1 + y)\ln(1 + \hat{y}) + (1 - y)\ln(1 - \hat{y})] + C$$

5. The gradient of this loss is

$$\frac{\partial \ell_{\text{logistic}}}{\partial w_i} = -(y - \hat{y})x_i, \quad \frac{\partial \ell_{\text{logistic}}}{\partial b} = -(y - \hat{y}),$$

   which exactly reproduces the Logistic Regression update rule under gradient descent.

Final Logistic Regression loss:

$$\boxed{\ell_{\text{logistic}}(w, b; x, y) = -\frac{1}{2}[(1 + y)\ln(1 + \tanh(b + w^T x)) + (1 - y)\ln(1 - \tanh(b + w^T x))]}$$

1.5: Conceptual Questions

Problem a

The softmax function is defined as,

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}.$$

It is called a *softargmax* because,

1. The argmax function outputs the index of the largest element in a vector, which is discrete and non-differentiable:
$$\text{argmax}(z) = \text{index of } \max_i z_i.$$
2. Softmax provides a differentiable approximation of argmax by converting a vector of values into a probability distribution that *weights larger values more heavily* while assigning smaller but nonzero probabilities to other entries. This makes it suitable for gradient-based optimization.
3. When one component of z is significantly larger than the others, softmax approaches a one-hot encoding of the argmax:
$$\text{softmax}(z) \approx \begin{cases} 1 & \text{for } i = \arg\max_j z_j, \\ 0 & \text{otherwise.} \end{cases}$$
4. This "*soft*" behavior allows the network to retain differentiability for backpropagation while still emphasizing the largest component, effectively preserving the ranking information of the maximum value.

Softmax is thus a differentiable, soft version of argmax, hence the term *softargmax*.

Problem b

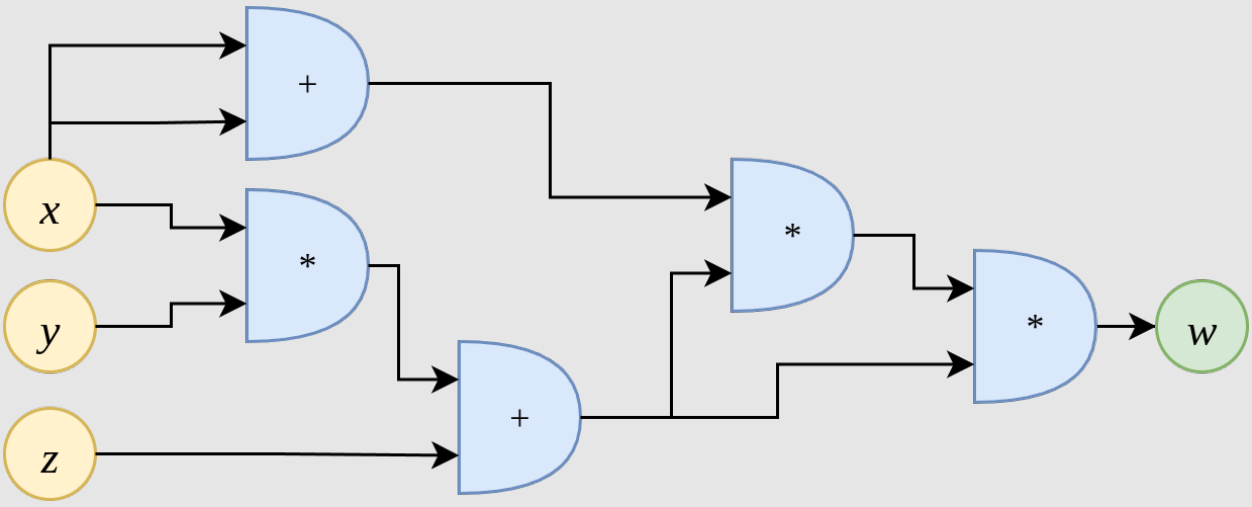

Figure 1: Computational graph for the function $a = x * y + z,\ b = (x + x) * a,\ w = a * b$ with scalar inputs $x, y, z \in \mathbb{R}$ and output $w \in \mathbb{R}$.

Problem c

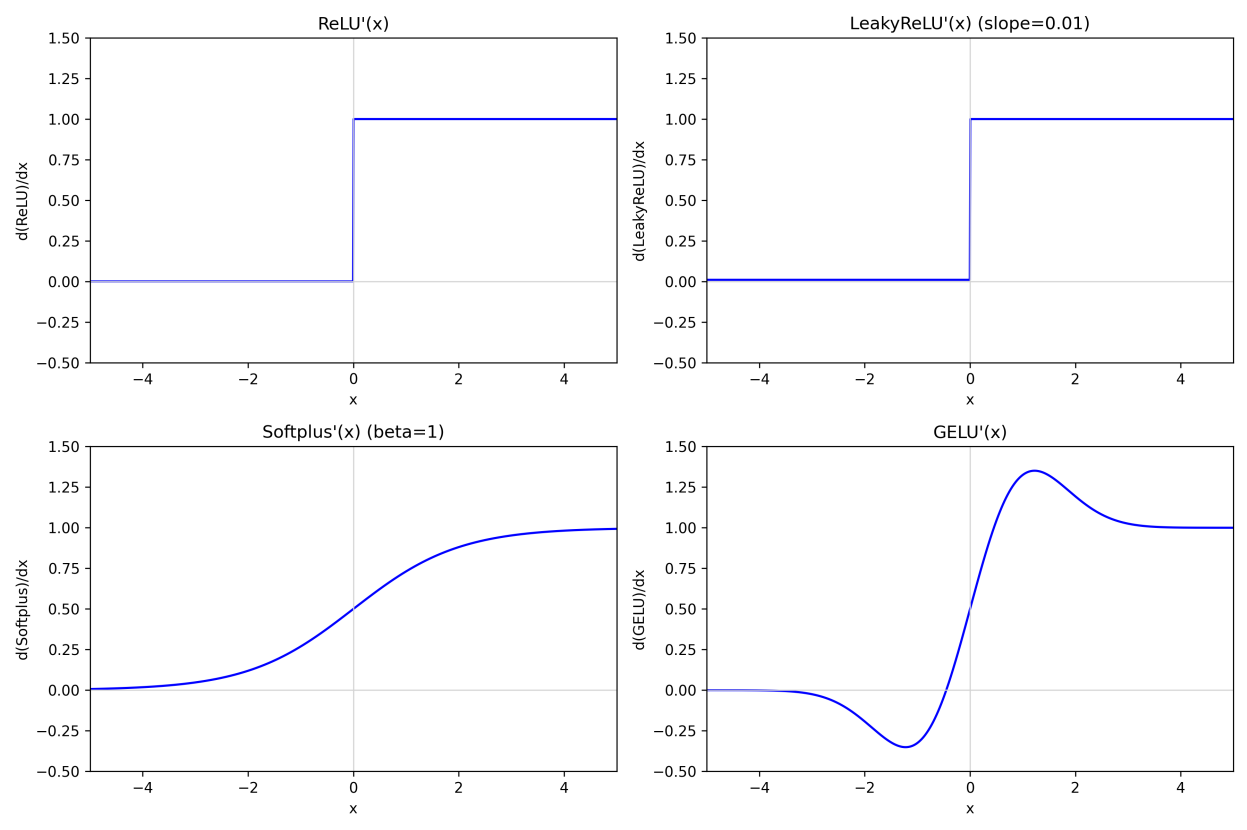

Figure 2: **Derivatives of activation functions**. Top-left: ReLU'(x), Top-right: LeakyReLU'(x) with slope 0.01, Bottom-left: Softplus'(x) with $\beta = 1$, Bottom-right: GELU'(x).

Problem d

Types of Linear Transformations
The four common types of linear transformations are,

1. **Scaling:** Scaling multiplies a vector x by a scalar factor α, changing its magnitude but keeping its direction unchanged. It either stretches or compresses the vector:

$$y = \alpha x, \quad \alpha \in \mathbb{R}$$

2. **Rotation:** Rotation transforms a vector in $\mathbb{R}^n$ without changing its length. It is represented by an orthogonal matrix $R \in \mathbb{R}^{n \times n}$, which preserves distances and angles:

$$y = Rx$$

3. **Reflection:** Reflection flips a vector over a line (in 2D) or a plane (in 3D). It is represented by a reflection matrix $F \in \mathbb{R}^{n \times n}$, which preserves distances but reverses orientation:

$$y = Fx$$

4. **Shearing:** Shearing shifts one coordinate of a vector proportionally to another, effectively slanting the shape. Shearing changes angles but preserves parallelism. The shear matrix $S \in \mathbb{R}^{n \times n}$ modifies vectors as:
$$y = Sx$$

Role of Linear and Non-Linear Transformations

Linear transformations in neural networks are applied via learnable weights W and bias b in each layer such that $h = Wx + b$. They serve to *combine* and *scale* input features, *capturing linear relationships* between them. Linear layers allow the network to rotate, scale, or project input data into a new feature space where patterns can become more separable. However, linear transformations alone *cannot capture complex, non-linear relationships*. No matter how many linear layers are stacked, the overall mapping from input to output remains linear. **Non-linear activation functions** $f(\cdot)$ are applied to the outputs of linear layers such that $a = f(h)$. These functions introduce *non-linearity*, enabling the network to *learn complex, non-linear mappings* from inputs to outputs. Non-linearities enable deep networks to learn hierarchical and abstract representations, which are essential for solving tasks where patterns are not linearly separable.

Problem e

Given a neural network F parameterized by θ, dataset $D = \{x_1, x_2, \ldots, x_N\}$, and labels $Y = \{y_1, y_2, \ldots, y_N\}$, training the network with the MSE loss is defined as minimizing the average squared difference between predictions and targets:

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

The training objective is to find parameters θ^* that minimize this loss:

$$\theta^* = \arg\min_\theta \mathcal{L}_{\text{MSE}}(\theta) = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \|F_\theta(x_i) - y_i\|^2.$$

1. **Forward pass:** For each input x_i, pass it through the neural network F_θ to compute the predicted output $\hat{y}_i$:
$$\hat{y}_i = F_\theta(x_i).$$

2. **Compute per-sample loss:** For each training example (x_i, y_i), compute the squared difference between the predicted value $\hat{y}_i$ and the true label y_i:
$$\ell_i = \|\hat{y}_i - y_i\|^2.$$

3. **Compute total loss:** Aggregate the per-sample losses over the entire dataset by taking the mean:
$$\mathcal{L}_{\text{MSE}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell_i = \frac{1}{N} \sum_{i=1}^{N} \|\hat{y}_i - y_i\|^2.$$
This gives the overall loss function that we aim to minimize during training, representing the average prediction error across all samples.

4. **Backward pass:** Perform backpropagation to compute the gradient of the total loss with respect to all network parameters θ:
$$\nabla_\theta \mathcal{L}_{\text{MSE}}(\theta).$$

5. **Parameter update:** Update the network parameters using gradient descent:

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

where η is the learning rate. This step adjusts the parameters in the direction that most reduces the loss, iteratively improving the network's predictions.

These steps are iterated over multiple epochs until convergence, i.e., until the parameters reach

$$\theta^* = \arg\min_\theta \mathcal{L}_{\text{MSE}}(\theta).$$