Three Questions and Their Detailed Solutions

Question 1: Backpropagation for a Three-Layer Network (ReLU + Linear + MAE)

Question Statement: *Show a backpropagation derivation for a three-layer network with a ReLU activation in the hidden layer, a linear activation at the output layer, and a Mean Absolute Error (MAE) loss. Derive the gradients of the loss w.r.t. all parameters.*

Solution/Derivation

Network Setup:

- Input: $\mathbf{x} \in \mathbb{R}^d$.
- Hidden layer with parameters $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$, $\mathbf{b}^{(1)} \in \mathbb{R}^h$, and ReLU activation.
- Output layer with parameters $\mathbf{W}^{(2)} \in \mathbb{R}^{1 \times h}$, $\mathbf{b}^{(2)} \in \mathbb{R}^1$, and *linear* activation.
- Loss: Mean Absolute Error (MAE), i.e. $\mathcal{L} = |\, a^{(2)} - y|$ where $a^{(2)}$ is the final output and y is the true label (scalar).

Forward Pass

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}, \qquad \mathbf{a}^{(1)} = \mathrm{ReLU}\big(\mathbf{z}^{(1)}\big),$$
$$z^{(2)} = \mathbf{W}^{(2)}\,\mathbf{a}^{(1)} + \mathbf{b}^{(2)}, \quad a^{(2)} = z^{(2)} \quad \text{(linear output)}.$$
$$\mathcal{L} = \big|\, a^{(2)} - y\big| = \big|\, z^{(2)} - y\,\big|.$$

Backprop Steps

1. Gradient of MAE w.r.t. $z^{(2)}$. For $\mathcal{L} = \big|z^{(2)} - y\big|$,

$$\frac{\partial \mathcal{L}}{\partial z^{(2)}} = \begin{cases} +1, & \text{if } z^{(2)} > y, \\ -1, & \text{if } z^{(2)} < y, \\ \text{(subgradient in } [-1, +1]), & \text{if } z^{(2)} = y. \end{cases}$$

We often write it as

$$\delta^{(2)} = \mathrm{sign}\big(z^{(2)} - y\big),$$

excluding the boundary case for simplicity.

2. Gradients w.r.t. $\mathbf{W}^{(2)}$ and $\mathbf{b}^{(2)}$.

$$z^{(2)} = \mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}.$$

Hence,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(2)}} = \delta^{(2)}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}} = \delta^{(2)} \left(\mathbf{a}^{(1)}\right)^{\top}.$$

3. Gradient w.r.t. hidden activations $\mathbf{a}^{(1)}$.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(1)}} = (\mathbf{W}^{(2)})^{\top}\, \delta^{(2)}.$$

4. ReLU derivative for $\mathbf{z}^{(1)}$.

$$\mathbf{a}^{(1)} = \mathrm{ReLU}(\mathbf{z}^{(1)}).$$

ReLU has derivative

$$\frac{d}{dz}\mathrm{ReLU}(z) = \begin{cases} 1, & z > 0, \\ 0, & z < 0, \end{cases}$$

with possible subgradient at $z = 0$. Let

$$\delta^{(1)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(1)}} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(1)}}\right) \odot \mathbf{1}_{(\mathbf{z}^{(1)}>0)},$$

where $\mathbf{1}_{(\mathbf{z}^{(1)}>0)}$ is elementwise 1 if $z_i^{(1)} > 0$, else 0.

5. Gradients w.r.t. $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$.

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\,\mathbf{x} + \mathbf{b}^{(1)}.$$

Thus,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(1)}} = \delta^{(1)}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}} = \delta^{(1)}\,\mathbf{x}^{\top}.$$

Final Summary of Gradients

$$\boxed{\begin{aligned} &\delta^{(2)} = \mathrm{sign}(z^{(2)} - y), \\ &\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(2)}} = \delta^{(2)}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}} = \delta^{(2)}\,(\mathbf{a}^{(1)})^{\top}, \\ &\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(1)}} = (\mathbf{W}^{(2)})^{\top}\,\delta^{(2)}, \quad \delta^{(1)} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(1)}}\right) \odot \mathbf{1}_{(\mathbf{z}^{(1)}>0)}, \\ &\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(1)}} = \delta^{(1)}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}} = \delta^{(1)}\,\mathbf{x}^{\top}. \end{aligned}}$$

Question 2: Group Normalization (GN) Explanation + Example

Question Statement: *Write in detail about Group Normalization, with an illustrative example. Compare it briefly to other normalizations (BN, LN), and show how the per-group statistics are computed in a CNN.*

Solution/Explanation

1. Motivation. Normalization methods improve training stability and speed by normalizing activations. *Batch Normalization* (BN) computes statistics across the mini-batch for each channel; *Layer Normalization* (LN) normalizes across all channels of a single sample. *Group Normalization* (GN) is in-between: it splits channels into G groups and normalizes within each group for each sample.

2. GN Formulas. Given a 4D feature map $x_{n,c,h,w}$ (batch n, channel c, spatial h, w), GN with G groups does:

(a) Partition C channels into G groups, each with $\frac{C}{G}$ channels.

(b) For each group g (within the same sample n), compute

$$\mu_g = \frac{1}{|\mathcal{S}_g|} \sum_{(c,h,w)\in\mathcal{S}_g} x_{n,c,h,w}, \quad \sigma_g^2 = \frac{1}{|\mathcal{S}_g|} \sum_{(c,h,w)\in\mathcal{S}_g} (x_{n,c,h,w} - \mu_g)^2,$$

where $\mathcal{S}_g$ collects the channels/spatial in group g.

(c) Normalize:

$$\hat{x}_{n,c,h,w} = \frac{x_{n,c,h,w} - \mu_g}{\sqrt{\sigma_g^2 + \epsilon}}.$$

(d) Learnable affine transform:

$$y_{n,c,h,w} = \gamma_c \, \hat{x}_{n,c,h,w} + \beta_c.$$

3. Why GN?

- **Small Batches:** BN can perform poorly when N is small. GN's statistics do not rely on N.
- **Flexibility:** Tuning G lets you trade off between LN-like ($G = 1$) or InstanceNorm-like ($G = C$).

4. Example. Suppose we have $N = 1$, $C = 8$, $H = 2$, $W = 2$, and choose $G = 2$ groups. Then each group has 4 channels. For *Group 1* (channels $\{1, 2, 3, 4\}$), we compute the mean/var across (4 channels) $\times$ (2×2 spatial) $= 16$ values. For *Group 2* (channels $\{5, 6, 7, 8\}$), we do likewise. Each group is normalized independently for that single sample.

5. Comparison with BN/LN.

- **BN:** Mean/var computed across *all samples in batch* for each channel. Works well with large batch sizes.
- **LN:** Mean/var computed across *all channels (and possibly spatial) of one sample.*
- **GN:** Splits channels into groups, computing mean/var within each group, for each sample. Mitigates large-batch reliance, typically stable for smaller batches, with lower memory overhead than LN if $G > 1$.

Thus, GN provides a robust alternative to BN in tasks or architectures where the batch dimension is small or irregular.

Question 3:

He vs. Xavier Initialization
Consider a fully connected layer with:

- **fan-in**: n_{in} (number of input units),
- **fan-out**: n_{out} (number of output units),
- weight matrix $\mathbf{W} \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$.

Suppose each weight W_{ij} is initialized as an i.i.d. Gaussian $\mathcal{N}(0, \sigma^2)$. Let $\mathbf{z} = \mathbf{W}\,\mathbf{x}$ be the pre-activation for a single layer, and $\mathbf{x}$ is a random vector with some variance $v_{\text{in}} = \text{Var}[x_i]$.

(a) **He Initialization (ReLU):**
Assume the activation function is ReLU: $\phi(z) = \max(0, z)$. Show that to maintain approximately constant variance of $\mathbf{z}$ and $\phi(\mathbf{z})$ across multiple layers, one typically chooses

$$\sigma^2 = \frac{2}{n_{\text{in}}}.$$

In your derivation, compute $\text{Var}[\phi(z)]$ given that z is zero-mean Gaussian with variance $\sigma^2\, n_{\text{in}}\, v_{\text{in}}$, and briefly explain why the factor of 2 arises for ReLU.

(b) **Xavier (Glorot) Initialization (tanh / sigmoid):**
Now assume a *saturating* activation such as tanh or $\sigma(\cdot)$. Show that if one balances the variance in both the forward pass (activations) *and* the backward pass (gradients), a more suitable variance choice is

$$\sigma^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}}.$$

State why this formula depends on both n_{in} and n_{out} and how it contrasts with the factor $\frac{2}{n_{\text{in}}}$ above.

(c) **Comparison:**
Summarize in one or two sentences the *key difference* between He vs. Xavier initialization—namely, *why* ReLU networks often use $\frac{2}{n_{\text{in}}}$ while saturating activations use $\frac{2}{n_{\text{in}}+n_{\text{out}}}$. (Optional: comment on how the formula changes for a leaky ReLU with slope $\alpha > 0$.)

(a) He Initialization for ReLU

Let $z = \sum_{i=1}^{n_{\text{in}}} w_i\, x_i$, where $w_i \sim \mathcal{N}(0, \sigma^2)$ and $\text{Var}[x_i] = v_{\text{in}}$. Then

$$\text{Var}[z] = \sum_{i=1}^{n_{\text{in}}} \text{Var}[w_i\, x_i] = \sum_{i=1}^{n_{\text{in}}} \sigma^2\, v_{\text{in}} = n_{\text{in}}\, \sigma^2\, v_{\text{in}}.$$

Assume z is approximately $\mathcal{N}(0,\, n_{\text{in}}\sigma^2\, v_{\text{in}})$.

Now consider $a = \phi(z) = \max(0, z)$. A standard result (or approximation) is that for $z \sim \mathcal{N}(0, \tau^2)$,

$$\text{Var}[\max(0, z)] \approx \frac{1}{2}\,\tau^2.$$

Thus,

$$\text{Var}[a] \approx \frac{1}{2}\, n_{\text{in}}\, \sigma^2\, v_{\text{in}}.$$

If we want $\text{Var}[a] \approx n_{\text{in}}\, \sigma^2\, v_{\text{in}}$ (or some constant scale not decaying or exploding), we see we require a factor of 2 in the denominator. In practice, it leads to

$$\sigma^2 = \frac{2}{n_{\text{in}}},$$

to keep the forward activations from decaying or exploding layer by layer when ϕ is ReLU.

(b) Xavier (Glorot) Initialization for Sigmoid/tanh

For a saturating nonlinearity (like sigmoid or tanh), the analysis typically seeks to preserve both:

- The forward variance of the layer's output,
- The backward variance of gradients (so that gradients neither explode nor vanish quickly).

Glorot & Bengio (2010) showed a balanced approach: if z is $\mathcal{N}(0, \sigma^2\, n_{\text{in}})$ in forward mode, then one obtains a consistent gradient flow if

$$\sigma^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}}.$$

When $n_{\text{in}} \approx n_{\text{out}}$, this is roughly $\frac{1}{n_{\text{in}}}$. The explicit appearance of n_{out} arises because the partial derivatives also depend on the number of outgoing connections in saturating networks, and balancing forward and backward signals across many layers is crucial.

(c) Comparison & Optional Extensions

In summary:

- **He init:** $\sigma^2 = \frac{2}{n_{\text{in}}}$ is best for ReLU (which zeroes out negatives). The factor of 2 helps maintain similar variance for the positive half of a Gaussian distribution.
- **Xavier init:** $\sigma^2 = \frac{2}{n_{\text{in}}+n_{\text{out}}}$ is more general for sigmoid/tanh, balancing variance of forward activations and backward gradients.

Leaky ReLU remark: If $\phi(z) = \max(\alpha z, z)$ for some $\alpha > 0$, one can refine the factor from 2 to a function of α (often between $1 + \alpha^2$ or related forms).

Question:4

Convexity of MSE and Cross-Entropy, and Their Importance

Q1. MSE Convexity:
Consider the Mean Squared Error (MSE) loss in the single-output case:

$$\mathcal{L}_{\text{MSE}}(y, \hat{y}) = \tfrac{1}{2}\,(\hat{y} - y)^2,$$

where $\hat{y} \in \mathbb{R}$ is the model prediction and $y \in \mathbb{R}$ is the true label.

(a) Show that, as a function of $\hat{y}$ (with y fixed), this loss is strictly convex. (Hint: second-derivative test)

(b) Why does strict convexity w.r.t. $\hat{y}$ *not* necessarily imply convexity w.r.t. the parameters θ of a neural network, where $\hat{y} = f_\theta(\mathbf{x})$?

Q2. Cross-Entropy Convexity:
Consider the **binary cross-entropy**:

$$\mathcal{L}_{\text{CE}}(y, p) = -\Big[y \ln(p) + (1 - y) \ln\big(1 - p\big)\Big],$$

where $y \in \{0, 1\}$ is the ground-truth label and $p \in (0, 1)$ is the predicted probability (e.g. via a sigmoid).

(a) Show that, as a function of p (with y fixed), $\mathcal{L}_{\text{CE}}$ is convex.

(b) Does the loss remain convex in p if you allow $y \in [0, 1]$ (a "soft" label)? Provide a short argument.

Q3. Importance of Convexity:

(a) Why might it matter (or not matter) that MSE and cross-entropy are convex w.r.t. $\hat{y}$ or p? How does this relate to gradient-based optimization?

(b) Given that neural networks rarely yield a convex loss *in parameters* θ, what benefit might we still gain from these losses being convex in *their predictions*?

1. MSE Convexity

(a) Let $f(\hat{y}) = \frac{1}{2}(\hat{y} - y)^2$. Then

$$\frac{df}{d\hat{y}} = (\hat{y} - y), \quad \frac{d^2 f}{d\hat{y}^2} = 1.$$

Since the second derivative is $1 > 0$, f is strictly convex in $\hat{y}$.

(b) Even though $f(\hat{y})$ is convex in $\hat{y}$, the map $\hat{y} = f_\theta(\mathbf{x})$ is typically nonlinear in parameters θ. In neural networks, f_θ can have many layers of composition, making $\hat{y}$ a non-convex function of θ. Thus, $\mathcal{L}(\theta) = f(f_\theta(\mathbf{x}))$ need *not* be convex in θ.

2. Cross-Entropy Convexity

(a) For binary cross-entropy:

$$g(p) = -\big[y \ln p + (1 - y) \ln(1 - p)\big], \quad y \in \{0, 1\}.$$

If $y = 1$, $g(p) = -\ln(p)$, which has second derivative $\frac{d^2}{dp^2}(-\ln p) = \frac{1}{p^2} > 0$ for $p > 0$. Similarly if $y = 0$, $g(p) = -\ln(1-p)$ also has a positive second derivative $\frac{1}{(1-p)^2}$. Hence $g(p)$ is convex in p.

(b) If $y \in [0, 1]$, we can write $g(p) = -\big[y \ln p + (1 - y) \ln(1 - p)\big]$ and still find

$$\frac{d^2 g}{dp^2} = \frac{y}{p^2} + \frac{1 - y}{(1 - p)^2},$$

which is > 0 for $p \in (0, 1)$. So it remains convex in p with $y \in [0, 1]$.

3. Importance

(a) **Why matter?** A convex loss in $\hat{y}$ or p ensures a *simple, bowl-shaped* landscape *if* we treat $\hat{y}, p$ as the optimization variables. This helps gradient-based algorithms converge reliably. However, in neural nets, the training typically optimizes w.r.t. θ, so *global* convexity in θ is not guaranteed.

(b) **Benefit in practice:** Even though the network's overall loss $\mathcal{L}(\theta)$ is non-convex, having a convex *per-sample* form in its *predictions* means local gradient signals are *straightforward*. For example, cross-entropy's gradient *does not saturate* as severely as other possible losses. This can still aid learning, even in a non-convex parameter space.