Assignment 2

November 25, 2025

1 Calculate $\nabla a^{[L]}(\mathbf{x})$

Network Definitions

The model is an L-layer feedforward neural network:

- **Input Layer** ($l = 1$): $\mathbf{a}^{[1]} = \mathbf{x} \in \mathbb{R}^{n_1}$
- **Linear Transformation** (For $l = 2, \ldots, L$):
$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$$
- **Activation** (For $l = 2, \ldots, L$):
$$\mathbf{a}^{[l]} = \sigma(\mathbf{z}^{[l]}) \in \mathbb{R}^{n_l}$$
- **Output Dimension**: $n_L = 1$ (Scalar output)

We aim to calculate the gradient of the scalar output $a^{[L]}$ with respect to the input vector $\mathbf{x}$, $\nabla a^{[L]}(\mathbf{x}) \in \mathbb{R}^{n_1}$.

Backpropagation Formulation

We define the sensitivity vector $\boldsymbol{\delta}^{[l]}$ as the gradient of the output $a^{[L]}$ with respect to the pre-activation vector $\mathbf{z}^{[l]}$. Since $a^{[L]}$ is a scalar, $\boldsymbol{\delta}^{[l]}$ is the row vector $\frac{\partial a^{[L]}}{\partial \mathbf{z}^{[l]}}$ viewed as a column vector.

$$\boldsymbol{\delta}^{[l]} = \left(\frac{\partial a^{[L]}}{\partial \mathbf{z}^{[l]}}\right)^T \in \mathbb{R}^{n_l}$$

1. Output Layer $\boldsymbol{\delta}^{[L]}$

For the output layer ($l = L$):

$$\boldsymbol{\delta}^{[L]} = \frac{\partial a^{[L]}}{\partial z^{[L]}} = \sigma'(z^{[L]})$$

(Since $n_L = 1$, $\mathbf{a}^{[L]}$ and $\mathbf{z}^{[L]}$ are scalars.)

2. Backpropagating $\boldsymbol{\delta}^{[l]}$

The relationship between $\boldsymbol{\delta}^{[l]}$ and $\boldsymbol{\delta}^{[l+1]}$ is derived using the Chain Rule:

$$\frac{\partial a^{[L]}}{\partial \mathbf{z}^{[l]}} = \frac{\partial a^{[L]}}{\partial \mathbf{z}^{[l+1]}} \cdot \frac{\partial \mathbf{z}^{[l+1]}}{\partial \mathbf{a}^{[l]}} \cdot \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{z}^{[l]}}$$

Where:

- $\frac{\partial a^{[L]}}{\partial \mathbf{z}^{[l+1]}} = (\boldsymbol{\delta}^{[l+1]})^T$
- $\frac{\partial \mathbf{z}^{[l+1]}}{\partial \mathbf{a}^{[l]}} = \mathbf{W}^{[l+1]}$
- $\frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{z}^{[l]}} = \operatorname{diag}(\sigma'(\mathbf{z}^{[l]}))$ (Diagonal matrix of derivatives)

Taking the transpose of the entire expression and simplifying:

$$\boldsymbol{\delta}^{[l]} = \text{diag}(\sigma'(\mathbf{z}^{[l]}))(\mathbf{W}^{[l+1]})^T \boldsymbol{\delta}^{[l+1]}$$

Using the property that $\text{diag}(\mathbf{v})\mathbf{u} = \mathbf{v} \circ \mathbf{u}$, where $\circ$ denotes the Hadamard (element-wise) product, we get the standard backpropagation step:

$$\boldsymbol{\delta}^{[l]} = \sigma'(\mathbf{z}^{[l]}) \circ \left((\mathbf{W}^{[l+1]})^T \boldsymbol{\delta}^{[l+1]}\right)$$

3. Final Gradient $\nabla a^{[L]}(\mathbf{x})$

The gradient $\nabla a^{[L]}(\mathbf{x})$ is obtained by relating $\boldsymbol{\delta}^{[2]}$ back to the input $\mathbf{x} = \mathbf{a}^{[1]}$.

$$\nabla a^{[L]}(\mathbf{x}) = \frac{\partial a^{[L]}}{\partial \mathbf{x}} = \frac{\partial a^{[L]}}{\partial \mathbf{z}^{[2]}} \cdot \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}}$$

Since $\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} = \mathbf{W}^{[2]}$ and $\frac{\partial a^{[L]}}{\partial \mathbf{z}^{[2]}} = (\boldsymbol{\delta}^{[2]})^T$, we have:

$$\nabla a^{[L]}(\mathbf{x}) = (\boldsymbol{\delta}^{[2]})^T \mathbf{W}^{[2]}$$

Taking the transpose to get the column vector $\nabla a^{[L]}(\mathbf{x}) \in \mathbb{R}^{n_1}$:

$$\nabla a^{[L]}(\mathbf{x}) = (\mathbf{W}^{[2]})^T \boldsymbol{\delta}^{[2]}$$

Algorithm for $\nabla a^{[L]}(\mathbf{x})$

Input and Output

- **Input**: Input vector $\mathbf{x} \in \mathbb{R}^{n_1}$, Network parameters $\{\mathbf{W}^{[l]}, \mathbf{b}^{[l]}\}_{l=2}^{L}$, Activation function $\sigma(\cdot)$.
- **Output**: Gradient $\nabla a^{[L]}(\mathbf{x}) \in \mathbb{R}^{n_1}$.

Forward Pass (Implicit)

(Must first compute all $\mathbf{z}^{[l]}$ and $\mathbf{a}^{[l]}$ for $l = 2, \ldots, L$ to get the required values.)

Backward Pass

1. **Step 1: Initialize $\boldsymbol{\delta}^{[L]}$ (Output Layer)**

$$\boldsymbol{\delta}^{[L]} = \sigma'(z^{[L]})$$

2. **Step 2: Backpropagate $\boldsymbol{\delta}^{[l]}$ (Hidden Layers)** For $l = L-1, L-2, \ldots, 2$:

$$\boldsymbol{\delta}^{[l]} = \sigma'(\mathbf{z}^{[l]}) \circ \left((\mathbf{W}^{[l+1]})^T \boldsymbol{\delta}^{[l+1]}\right)$$

   (Compute this down to $\boldsymbol{\delta}^{[2]}$)

3. **Step 3: Calculate Final Gradient**

$$\nabla a^{[L]}(\mathbf{x}) = (\mathbf{W}^{[2]})^T \boldsymbol{\delta}^{[2]}$$

---

2 General Questions on Network Architecture

Q1: Does having more layers or more neurons in a neural network lead to a better result?

Adding more layers (**depth**) or more neurons per layer (**width**) both increase the **capacity** (expressive power) of a neural network. However, whether this leads to a "better result" depends crucially on the task complexity, dataset size, and the optimization process.

1. **More Layers (Depth)**
   - **Pros**: Enables the network to capture complex, **hierarchical features** (e.g., edges $\rightarrow$ shapes $\rightarrow$ objects in image data). Deep networks often require fewer parameters than wide shallow networks to achieve the same expressive power.
   - **Cons**: Harder to train due to issues like **vanishing/exploding gradients**. Higher computational cost for the forward and backward passes.

2. **More Neurons (Width)**
   - **Pros**: Increases the immediate expressive power of a single layer; generally **easier to train** and less susceptible to the extreme vanishing gradient problem compared to very deep networks.
   - **Cons**: Leads to a much larger number of parameters, increasing the risk of **overfitting** if data is scarce. Higher memory consumption.

3. **The Right Model Size**

The goal is to find the architecture that best fits the complexity of the data without memorizing the noise.

- **Underfitting (Too Small)**: If the model has too few parameters (low capacity), it fails to learn the underlying patterns.
- **Overfitting (Too Large)**: If the model has too many parameters (high capacity), it learns the training data and noise too well, leading to poor generalization on unseen data.

4. **Key Idea: Diagnosis-Based Adjustments**
   - If your model is **underfitting** (poor performance on both train and test sets), you should generally **increase size** (more depth or width) or train longer.
   - If your model is **overfitting** (good performance on train, poor on test), you should **reduce size**, or more commonly, apply stronger **regularization** (L1/L2, Dropout) or acquire **more data**.
   - If training is **unstable** (loss fluctuates wildly or doesn't converge), the depth might be too large, requiring advanced techniques (e.g., Residual Connections, Batch Normalization).