Introduction to Deep Learning

Mnacho Echenim

Grenoble INP-Ensimag

2022-2023

Notes

Evaluating how well an MLP is doing (supervised learning)

Problem we have a set of samples $S = \left\{(\alpha^0_{(i)}, \rho_{(i)}) \,\middle|\, i = 1, \dots, N\right\}$ and an MLP parameterized by a set of weights and biases collectively denoted by θ

Goal find θ^* such that, for all $i = 1, \dots, N$, when the input neurons are fed x_i, the output activation $\widehat{\rho_{(i)}}$ is a **good** approximation of $\rho_{(i)}$

Definition

Cost function: positive function $\mathcal{E}(S, \theta)$ meant to measure how well an MLP is doing

Standard assumption

The cost function can be written as an average of individual cost functions over all samples: $\mathcal{E}(S, \theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{C}(\theta, \rho_{(i)}, \widehat{\rho_{(i)}})$

Reformulation

Goal: solve the optimization problem $\theta^* = \arg\min_\theta \mathcal{E}(S, \theta)$

Notes

Examples of individual cost functions

- Mean square error (MSE) $(y, \widehat{y}) \mapsto \frac{1}{2} \cdot (y - \widehat{y})^2$
- L_1 error $(y, \widehat{y}) \mapsto |y - \widehat{y}|$ (robust regression)
- $(y, \widehat{y}) \mapsto \log(1 + \exp(-y\widehat{y}))$ (logistic regression)
- $(y, \widehat{y}) \mapsto \max(0, -y\widehat{y})$ (binary classification)

Notes

Gradient descent

Goal: find the minimum of $\mathcal{E}(S, \theta)$ using gradient descent: if θ_k denotes the network parameters at iteration k then

$$\theta_{k+1} \leftarrow \theta_k - \eta \nabla_\theta \mathcal{E}(S, \theta_k)$$

The **hyperparameter** η is called the **learning rate**. In our case, the parameter update can be written as:

$$\theta_{k+1} \leftarrow \theta_k - \frac{\eta}{N} \sum_{i=1}^{N} \nabla_\theta \mathcal{C}(\theta_k, \rho_{(i)}, \widehat{\rho_{(i)}})$$

Issues:

- Computing a single gradient can be very long
- Vectorization is not possible for large samples

Notes

Stochastic gradient descent

- **Idea:** why not use the gradient from a single **arbitrary** sample?
  The update rule becomes $\theta_{k+1} \leftarrow \theta_k - \eta \cdot \nabla_\theta \mathcal{C}(\theta_k, \rho_{(i_k)}, \widehat{\rho_{(i_k)}})$, where i_k is a random variable that follows the discrete uniform distribution on $\{1, \ldots, N\}$
- **Why random?**
- **Features**
  - Faster to compute
  - Can be used for online learning
  - Can avoid local minima, saddle points (more on this later)

Notes

Features of SGD

- Convergence can be much slower than for gradient descent
- Problems arise when we are close to the optimal value θ^*: the noise becomes problematic
- Can this noise be reduced?

Notes

A trade-off: Mini-batch gradient descent

- **Principle:** use M arbitrary samples to optimize cost function

  The update rule becomes

$$\theta_{k+1} \leftarrow \theta_k - \frac{\eta}{M} \sum_{j=1}^{M} \nabla_\theta \mathcal{C}(\theta_k, \rho_{(i_{k_j})}, \widehat{\rho_{(i_{k_j})}}),$$

  where i_{k_j} is a random variable that follows the discrete uniform distribution on $\{1, \ldots, N\}$

- Features
  - Less noisy approximation of real gradient
  - Computation cost can be controlled
    - Mini-batch size

Notes

Summary

Given an MLP and a sample set of size N:

Method	Updates per epoch	Computations per update

Notes

About backpropagation

- Computation of gradients during the training phase
  - How should weights and biases be updated to take into account that the output of the network is not correct?
  - Very efficient
  - One of the reasons deep neural networks are so successful
- A special case of automatic differentiation
  - Generally presented with **computational graphs**
  - Modern ML frameworks implement the general case
  - Here, we focus on the derivation of backpropagation equations for neural networks

Notes

A word of warning

- The goal of this part is to **derive** the backpropagation rules, not simply verify that they work
- The presentation will be quite formal, to ensure each step is well understood
- This makes notations quite heavy
- An interesting exercise is to try to derive the rules by yourselves, using the lighter notations that are more standard

Notes

Some definitions

Definition

Given the matrices $M_1, \ldots, M_p$, we define the bloc matrix

$$\mathrm{diag}\,(M_1, \ldots, M_p) \stackrel{\text{def}}{=} \begin{pmatrix} M_1 & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & M_2 & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & M_p \end{pmatrix}$$

Definition

Given $A, B \in \mathbb{R}^{n \times m}$, the **Hadamard product** of A and B, denoted by $A \odot B$, is the matrix such that $(A \odot B)_{i,j} = A_{i,j} \cdot B_{i,j}$ (pointwise multiplication)

Notes

The chain rule

Definition

The **Jacobian matrix** of a function $\begin{array}{rccc} f: & \mathbb{R}^n & \to & \mathbb{R}^m \\ & a & \mapsto & (f_1(a), \ldots, f_m(a))^{\mathrm{T}} \end{array}$

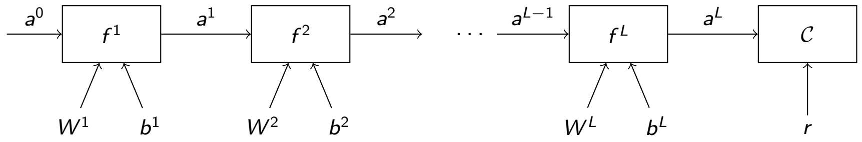
is the function that associates to $\alpha \in \mathbb{R}^n$ the matrix in $\mathbb{R}^{m \times n}$ denoted by $\frac{\partial f}{\partial a}(\alpha)$, where $\left(\frac{\partial f}{\partial a}(\alpha)\right)_{i,j} = \frac{\partial f_i}{\partial a_j}(\alpha)$

Proposition

Given: $\begin{array}{rccc} f: & \mathbb{R}^n & \to & \mathbb{R}^m \\ & a & \mapsto & f(a) \end{array}$ $\quad \begin{array}{rccc} g: & \mathbb{R}^m & \to & \mathbb{R}^p \\ & y & \mapsto & g(y) \end{array}$ $\quad \begin{array}{rccc} h: & \mathbb{R}^n & \to & \mathbb{R}^p \\ & a & \mapsto & g \circ f(a) \end{array}$

Notes

Some formalism



Where for $j = 1, \dots, L$:

- $a^{j-1} \in \mathbb{R}^{n_{j-1}}$ represents the formal input of layer j
- $b^j \in \mathbb{R}^{n_j}$ and $W^j = (w_1^j, \dots, w_{n_j}^j)$, where for $k = 1, \dots, n_j$, $w_k^j \in \mathbb{R}^{n_{j-1}}$ represents the weights for neuron k at layer j
- f^j represents the computation performed at layer j:

$$\begin{aligned} f^j\colon\ (\mathbb{R}^{n_{j-1}})^{n_j+1} \times \mathbb{R}^{n_j} &\to \mathbb{R}^{n_j} \\ a^{j-1}, W^j, b^j &\mapsto f^j(a^{j-1}, W^j, b^j) \end{aligned}$$

$\mathcal{C}$ is the error function:
$$\begin{aligned} \mathcal{C}\colon \mathbb{R}^{n_L} \times \mathbb{R}^{n_L} &\to \mathbb{R}^+ \\ a^L, r &\mapsto \mathcal{C}(a^L, r) \end{aligned}$$

Notes

Reminders

- Parameters

Name	Formal	Actual	Dimension
Activation of layer j	a^j	α^j	$\mathbb{R}^{n_j}$
Weights at layer j	W^j	$\Omega^j = (\omega_1^j, \dots, \omega_{n_j}^j)$	$\mathbb{R}^{n_{j-1} \times n_j}$
Bias at layer j	b^j	β^j	$\mathbb{R}^{n_j}$
Expected output	r	ρ	$\mathbb{R}$

- Partial derivatives

$$\begin{aligned} \frac{\partial f^j}{\partial a^{j-1}}(\alpha^{j-1}, \Omega^j, \beta^j) &\in \mathbb{R}^{n_j \times n_{j-1}} \\ \frac{\partial f^j}{\partial w_k^j}(\alpha^{j-1}, \Omega^j, \beta^j) &\in \mathbb{R}^{n_j \times n_{j-1}}, \text{ for } k = 1, \dots, n_j \\ \frac{\partial f^j}{\partial b^j}(\alpha^{j-1}, \Omega^j, \beta^j) &\in \mathbb{R}^{n_j \times n_j} \end{aligned}$$

Notes

What we want to compute

Let g^i represent the computation of the first i layers of the network:

Let $\mathcal{E}$ represent the output of the error function:

Output of i^{th} layer of the network, given the inputs α^0, Ω^1, β^1, $\ldots$, Ω^i, β^i:

Error of the network when expected result is ρ:

Goal

Compute $\frac{\partial \mathcal{E}}{\partial w_k^j}(\alpha^0, \Omega^1, \beta^1, \ldots, \Omega^L, \beta^L, \rho)$ and $\frac{\partial \mathcal{E}}{\partial b^j}(\alpha^0, \Omega^1, \beta^1, \ldots, \Omega^L, \beta^L, \rho)$

Notes

Using the chain rule

Chain rule on the error function:

$$\frac{\partial \mathcal{E}}{\partial w_k^j}(\alpha^0, \ldots, \Omega^L, \beta^L, \rho) = \frac{\partial \mathcal{C}}{\partial a^L}(\alpha^L, \rho) \cdot \frac{\partial g^L}{\partial w_k^j}(\alpha^0, \ldots, \Omega^L, \beta^L)$$

$$\frac{\partial \mathcal{E}}{\partial b^j}(\alpha^0, \ldots, \Omega^L, \beta^L, \rho) = \frac{\partial \mathcal{C}}{\partial a^L}(\alpha^L, \rho) \cdot \frac{\partial g^L}{\partial b^j}(\alpha^0, \ldots, \Omega^L, \beta^L)$$

Chain rule on the output of the network when $j = L$:

$$\frac{\partial g^L}{\partial w_k^L}(\alpha^0, \ldots, \Omega^L, \beta^L) = \frac{\partial f^L}{\partial w_k^L}(\alpha^{L-1}, \Omega^L, \beta^L)$$

$$\frac{\partial g^L}{\partial b^L}(\alpha^0, \ldots, \Omega^L, \beta^L) = \frac{\partial f^L}{\partial b^L}(\alpha^{L-1}, \Omega^L, \beta^L)$$

Notes

Using the chain rule (2)

Chain rule on the output of the network when $j < L$:

$$\begin{aligned}
\frac{\partial g^L}{\partial w_k^j}(\alpha^0, \ldots, \Omega^L, \beta^L) &= \left[\prod_{i=L}^{j+1} \frac{\partial f^i}{\partial a^{i-1}}(\alpha^{i-1}, \Omega^i, \beta^i)\right] \cdot \frac{\partial f^j}{\partial w_k^j}(\alpha^{j-1}, \Omega^j, \beta^j) \\
\frac{\partial g^L}{\partial b^j}(\alpha^0, \ldots, \Omega^L, \beta^L) &= \left[\prod_{i=L}^{j+1} \frac{\partial f^i}{\partial a^{i-1}}(\alpha^{i-1}, \Omega^i, \beta^i)\right] \cdot \frac{\partial f^j}{\partial b^j}(\alpha^{j-1}, \Omega^j, \beta^j)
\end{aligned}$$

Notes

Recap

Consider the inputs $\alpha^0, \Omega^1, \beta^1, \ldots, \Omega^L, \beta^L$ and the expected result ρ, and let

$$\mathcal{P}^{L+1} \stackrel{\text{def}}{=} \frac{\partial \mathcal{C}}{\partial a^L}(\alpha^L, \rho) \qquad \mathcal{P}^j \stackrel{\text{def}}{=} \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial a^{j-1}}(\alpha^{j-1}, \Omega^j, \beta^j)$$

Then for $j = 1, \ldots, L$, we have $\mathcal{P}^j \in \mathbb{R}^{1 \times n_{j-1}}$ and

$$\frac{\partial \mathcal{E}}{\partial w_k^j}(\alpha^0, \ldots, \Omega^L, \beta^L, \rho) = \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial w_k^j}(\alpha^{j-1}, \Omega^j, \beta^j)$$

$$\frac{\partial \mathcal{E}}{\partial b^j}(\alpha^0, \ldots, \Omega^L, \beta^L, \rho) = \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial b^j}(\alpha^{j-1}, \Omega^j, \beta^j)$$

At layer j, we receive $\mathcal{P}^{j+1}$ and we need to compute $\frac{\partial f^j}{\partial a^{j-1}}(\alpha^{j-1}, \Omega^j, \beta^j)$, $\frac{\partial f^j}{\partial w_k^j}(\alpha^{j-1}, \Omega^j, \beta^j)$ and $\frac{\partial f^j}{\partial b^j}(\alpha^{j-1}, \Omega^j, \beta^j)$

Notes

Same recap, with simplified notations

Consider the inputs $\alpha^0, \Omega^1, \beta^1, \ldots, \Omega^L, \beta^L$ and the expected result ρ, and let

$$\mathcal{P}^{L+1} \stackrel{\text{def}}{=} \frac{\partial \mathcal{C}}{\partial a^L} \qquad \mathcal{P}^j \stackrel{\text{def}}{=} \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial a^{j-1}}$$

Then for $j = 1, \ldots, L$, we have $\mathcal{P}^j \in \mathbb{R}^{1 \times n_{j-1}}$ and

$$\frac{\partial \mathcal{E}}{\partial w_k^j} = \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial w_k^j}$$

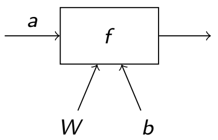
$$\frac{\partial \mathcal{E}}{\partial b^j} = \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial b^j}$$

At layer j, we need to compute $\frac{\partial f^j}{\partial a^{j-1}}$, $\frac{\partial f^j}{\partial w_k^j}$ and $\frac{\partial f^j}{\partial b^j}$

Notes

Partial derivatives for a single layer

For a layer with n inputs
and m neurons we have the following:



- $a \in \mathbb{R}^n$
- $b = (b_1, \ldots, b_m)^{\mathrm{T}}$
  and $W = (w_1, \ldots, w_m)$,
  where for $p = 1, \ldots, m$, $w_p \in \mathbb{R}^n$
- f is defined by:

$$\begin{aligned} f\colon\ (\mathbb{R}^n)^{m+1} \times \mathbb{R}^m &\to \mathbb{R}^m \\ a, W, b &\mapsto [h(a, w_1, b_1), \ldots, h(a, w_m, b_m)]^{\mathrm{T}} \end{aligned}$$

- $h = \Phi \circ \Psi$, where

$$\begin{aligned} \Phi\colon \mathbb{R} &\to \mathbb{R} \\ z &\mapsto \Phi(z) \end{aligned} \qquad \text{and} \qquad \begin{aligned} \Psi\colon \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} &\to \mathbb{R} \\ a, w, b &\mapsto w^{\mathrm{T}} a + b \end{aligned}$$

Notes

Partial derivatives for a single neuron

- Input: α (neuron input), β (bias) and ω (weights for the neuron)
  - Let ζ denote the net input of the neuron
  - $\zeta = \omega^{\mathrm{T}}\alpha + \beta = \Psi(\alpha, \omega, \beta)$
- Output: $h(\alpha, \omega, \beta) = \Phi(\Psi(\alpha, \omega, \beta)) = \Phi(\zeta) = \Phi(\omega^{\mathrm{T}}\alpha + \beta)$
- We have:

$$\frac{\partial h}{\partial a}$$

$$\frac{\partial h}{\partial w}$$

$$\frac{\partial h}{\partial b}$$

Notes

Back to partial derivatives for a layer

$f : a, W, b \mapsto [h(a, w_1, b_1), \dots, h(a, w_m, b_m)]^{\mathrm{T}}$
Input α, Ω, β, where $\Omega = (\omega_1, \dots, \omega_m)$

$$\frac{\partial f}{\partial a} = \begin{pmatrix} \Phi'(\zeta_1).\omega_1^{\mathrm{T}} \\ \vdots \\ \Phi'(\zeta_m).\omega_m^{\mathrm{T}} \end{pmatrix}$$

$$\frac{\partial f}{\partial w_k} = \begin{pmatrix} 0 \\ \vdots \\ \Phi'(\zeta_k).\alpha^{\mathrm{T}} \\ \vdots \\ 0 \end{pmatrix} \leftarrow \text{ line } k$$

$$\frac{\partial f}{\partial b_k} = (0, \dots, \Phi'(\zeta_k), \dots, 0)^{\mathrm{T}}$$

$$\frac{\partial f}{\partial b} = \mathrm{diag}\left(\Phi'(\zeta_1), \dots, \Phi'(\zeta_m)\right)$$

Notes

Back to partial derivatives for an entire network

$$\mathcal{P}^{L+1} = \frac{\partial \mathcal{C}}{\partial a^L} \quad \text{and} \quad \mathcal{P}^j = \mathcal{P}^{j+1} \cdot \begin{pmatrix} \Phi'(\zeta_1^j).\left[\omega_1^j\right]^{\mathrm{T}} \\ \vdots \\ \Phi'(\zeta_{n_j}^j).\left[\omega_{n_j}^j\right]^{\mathrm{T}} \end{pmatrix}$$

$$\frac{\partial \mathcal{E}}{\partial w_k^j} = \mathcal{P}^{j+1} \cdot \begin{pmatrix} 0 \\ \vdots \\ \Phi'(\zeta_k^j).\left[\alpha^{j-1}\right]^{\mathrm{T}} \\ \vdots \\ 0 \end{pmatrix} \leftarrow \text{line } k$$

$$\frac{\partial \mathcal{E}}{\partial b^j} = \mathcal{P}^{j+1} \cdot \operatorname{diag}\left(\Phi'(\zeta_1^j), \ldots, \Phi'(\zeta_{n_j}^j)\right)$$

Notes

Back to gradient descent

- We are interested in computing
  - $\nabla_{w_k^j} \mathcal{E} \stackrel{\text{def}}{=} \left[\frac{\partial \mathcal{E}}{\partial w_k^j}\right]^{\mathrm{T}}$
  - $\nabla_{b^j} \mathcal{E} \stackrel{\text{def}}{=} \left[\frac{\partial \mathcal{E}}{\partial b^j}\right]^{\mathrm{T}}$
- Weights are stored in a matrix: $\Omega^j = (\omega_1^j, \ldots, \omega_{n_j}^j) \in \mathbb{R}^{n_{j-1} \times n_j}$
- If we define

$$\nabla_{W^j} \mathcal{E} \stackrel{\text{def}}{=} \begin{pmatrix} \nabla_{w_1^j} \mathcal{E} & \cdots & \nabla_{w_{n_j}^j} \mathcal{E} \end{pmatrix}$$

- Then parameters updates for stochastic gradient descent become

$$\begin{aligned} \Omega^j &\leftarrow \Omega^j - \eta \cdot \nabla_{W^j} \mathcal{E} \\ \beta_j &\leftarrow \beta_j - \eta \cdot \nabla_{b^j} \mathcal{E} \end{aligned}$$

Notes

Effective computations

$$\mathcal{P}^j = \mathcal{P}^{j+1}\cdot\begin{pmatrix}\Phi'(\zeta_1^j).\left[\omega_1^j\right]^{\mathrm{T}}\\ \vdots \\ \Phi'(\zeta_{n_j}^j).\left[\omega_{n_j}^j\right]^{\mathrm{T}}\end{pmatrix} = \mathcal{P}^{j+1}\cdot\mathrm{diag}\left(\Phi'(\zeta_1^j),\ldots,\Phi'(\zeta_{n_j}^j)\right)\cdot\left[\Omega^j\right]^{\mathrm{T}}$$

$$= \left[\Phi'(\zeta^j)\odot\left[\mathcal{P}^{j+1}\right]^{\mathrm{T}}\right]^{\mathrm{T}}\cdot\left[\Omega^j\right]^{\mathrm{T}}$$

$$\nabla_{W^j}\mathcal{E} = \left(\cdots \quad \underbrace{\left(0,\cdots,0,\underbrace{\alpha^{j-1}.\Phi'(\zeta_k^j)}_{\text{column } k},0,\cdots,0\right)\left[\mathcal{P}^{j+1}\right]^{\mathrm{T}}}_{k^{\text{th}}\text{ vector}} \quad \cdots\right)$$

$$= \left(\alpha^{j-1}.\left(\Phi'(\zeta_1^j)\cdot\mathcal{P}_1^{j+1}\right) \quad \cdots \quad \alpha^{j-1}.\left(\Phi'(\zeta_{n_j}^j)\cdot\mathcal{P}_{n_j}^{j+1}\right)\right)$$

$$= \alpha^{j-1}\cdot\left[\Phi'(\zeta^j)\odot\left[\mathcal{P}^{j+1}\right]^{\mathrm{T}}\right]^{\mathrm{T}}$$

$$\nabla_{b^j}\mathcal{E} = \Phi'(\zeta^j)\odot\left[\mathcal{P}^{j+1}\right]^{\mathrm{T}}$$

Notes

Backpropagation rules

Let $\mathcal{B}^j \stackrel{\text{def}}{=} \Phi'(\zeta^j)\odot\left[\mathcal{P}^{j+1}\right]^{\mathrm{T}} \in \mathbb{R}^{n_j\times 1}$ for $j = 1,\ldots,L-1$, so that

$$\mathcal{P}^j = \left[\Phi'(\zeta^j)\odot\left[\mathcal{P}^{j+1}\right]^{\mathrm{T}}\right]^{\mathrm{T}}\cdot\left[\Omega^j\right]^{\mathrm{T}} = \left[\mathcal{B}^j\right]^{\mathrm{T}}\cdot\left[\Omega^j\right]^{\mathrm{T}} = \left[\Omega^j\mathcal{B}^j\right]^{\mathrm{T}}$$

Recall also that $\mathcal{P}^{L+1} = \frac{\partial\mathcal{C}}{\partial a^L}$

Notes

Backpropagation and gradient computation

Input: A network with L layers
Input: α^0 that has been forward propagated
Input: ρ the expected output

$\mathcal{B}^L \leftarrow \Phi'(\zeta^L) \odot \nabla_{a^L}\mathcal{C}(\alpha^L, \rho)$;
$\left[\mathcal{P}^L\right]^{\mathrm{T}} \leftarrow \Omega^L \cdot \mathcal{B}^L$;
for $j \leftarrow L - 1$ **to** 1 **do**
    $\mathcal{B}^j \leftarrow \Phi'(\zeta^j) \odot \left[\mathcal{P}^{j+1}\right]^{\mathrm{T}}$;
    $\left[\mathcal{P}^j\right]^{\mathrm{T}} \leftarrow \Omega^j \cdot \mathcal{B}^j$;
end

Algorithm 1: Backpropagation algorithm

Input: A network with L layers
Input: α^0 that has been forward propagated
Input: $(\mathcal{B}^1, \ldots, \mathcal{B}^L)$ that have been updated by backpropagation

for $j \in \{1, \ldots, L\}$ **do**
    $\mathrm{Gradient}(\Omega^j) \leftarrow \alpha^{j-1} \cdot \left[\mathcal{B}^j\right]^{\mathrm{T}}$;
    $\mathrm{Gradient}(\beta^j) \leftarrow \mathcal{B}^j$;
end

Algorithm 2: Gradient computation

Notes

Bonus: verifying that gradients are correctly computed

Input: Network N with L layers
Input: α^0, ρ
Input: ε, δ

N.Propagate(α^0);
N.Learn(N.Output, ρ);
for $l \in \{1, \ldots, L\}$ **do**
    let Ω^l be the weight matrix for layer l;
    for $(i, j) \in [\![1, n_l]\!] \times [\![1, m_l]\!]$ **do**
        let $\omega_{i,j} = \Omega^l[i,j]$ **and** $\omega^+_{i,j} = \omega_{i,j} \cdot (1 + \varepsilon)$ **and** $\omega^-_{i,j} = \omega_{i,j} \cdot (1 - \varepsilon)$;
        let $N^+ = N\left[\Omega^l[i,j] \leftarrow \omega^+_{i,j}\right]$ **and** $N^- = N\left[\Omega^l[i,j] \leftarrow \omega^-_{i,j}\right]$;
        N^+.Propagate(α^0);
        N^-.Propagate(α^0);
        let $C^+ = \mathcal{C}(N^+.\mathrm{Output}, \rho)$ **and** $C^- = \mathcal{C}(N^-.\mathrm{Output}, \rho)$;
        let $A^l_{i,j} = \frac{C^+ - C^-}{2\varepsilon \cdot \omega_{i,j}}$;
        let $G^l_{i,j} = N.\mathrm{Gradient}(\Omega^l)[i,j]$;
        Assert $\frac{|A^l_{i,j} - G^l_{i,j}|}{|A^l_{i,j}| + |G^l_{i,j}|} \leq \delta$
    end
end

Notes