Foundations of Data Science, Fall 2020

13. Neural Networks I

Prof. Dan Olteanu

DaST Data•(Systems+Theory)

University of Zurich[UZH]

Nov 17, 2020

https://lms.uzh.ch/url/RepositoryEntry/16830890400

https://uzh.zoom.us/j/96690150974?pwd=cnZmMTduWUtCeWoxYW85Z3RMYnpTZz09

Outline

Today, we'll study feedforward neural networks

- Multi-layer perceptrons
- Classification or regression settings
- Backpropagation to compute gradients

Recall: The Perceptron

Perceptron

- Input: Feature vector $\mathbf{x}$ and label y
- Output: $\operatorname{sign}(b + \mathbf{w} \cdot \mathbf{x})$, where $b =$ bias term, $\mathbf{w} =$ weight parameters

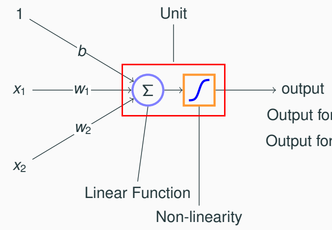
Artificial neuron (unit): generalisation of perceptron

- Activation function $f : \mathbb{R} \to \mathbb{R}$
- Output: $f(b + \mathbf{w} \cdot \mathbf{x})$
- Can be used to build Boolean gates (recall Exercise Sheet 2)

Neural Networks: Composition of artificial neurons into networks

- Can compute any function that a computer can
- The units have continuous activation functions
- Suitably chosen loss function for any input-output is a differentiable function

Neural Network with one Artificial Neuron



Output for regression: $\widehat{y} = \mathbb{E}[y \mid \mathbf{x}, \mathbf{W}, \mathbf{b}]$

Output for classification: $\widehat{y} = \Pr(y = 1 \mid \mathbf{x}, \mathbf{w}, \mathbf{b})$

A unit in a neural network computes a linear function of its input and is then composed with a non-linear activation function

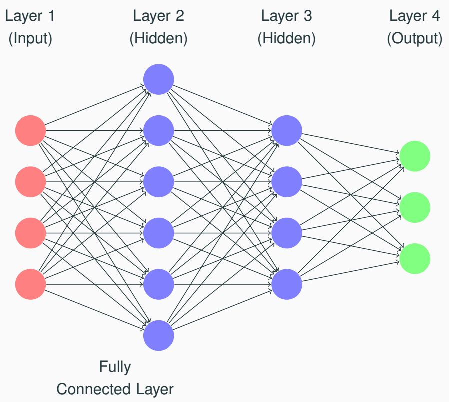
For logistic regression

- The non-linear activation function is the sigmoid

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

- The separating surface is linear

Feedforward Neural Networks



Units are organised in a layered directed-acyclic graph (no loops).

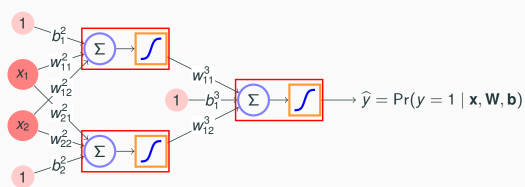
Multilayer Perceptron (MLP)

Multilayer Perceptron

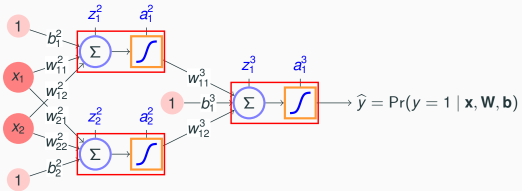
- Feedforward neural network
- The units do not have to be perceptrons

Three-layered MLP for classification using sigmoid activation functions:

- First maps the feature vector $\mathbf{x}$ to non-linear features
- Then applies logistic regression to compute $\widehat{y}$



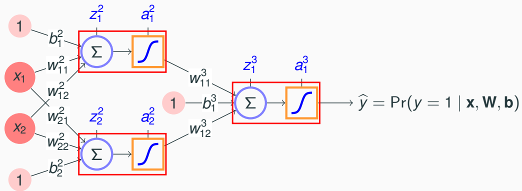
MLP: Terminology and Notation



Layers: (1) Input; (2) Hidden, with two units; (3) Output, here classification

- w_{ij}^{ℓ}: weight for the connection to unit i in layer ℓ from unit j in layer $\ell - 1$
- b_i^{ℓ}: bias term for unit i in layer ℓ
- z_i^{ℓ}: pre-activation for unit i in layer ℓ, sum of weighted outputs from $\ell - 1$
- a_i^{ℓ}: activation for unit i in layer ℓ, non-linear function of pre-activation

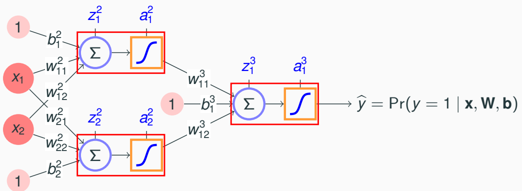
MLP: Matrix Notation (Layer 1)



Layer 1: Pre-activation = activation = input feature vector

$$\mathbf{a}^1 = \mathbf{z}^1 = \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

MLP: Matrix Notation (Layer 2)

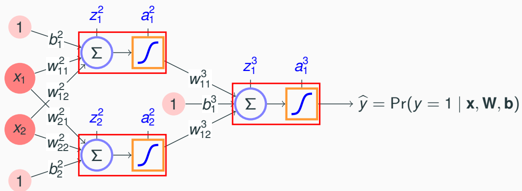


Weights in row vectors $\mathbf{W}^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \end{bmatrix}$ Bias term vector $\mathbf{b}^2 = \begin{bmatrix} b_1^2 \\ b_2^2 \end{bmatrix}$

Pre-activation $\mathbf{z}^2 = \begin{bmatrix} z_1^2 \\ z_2^2 \end{bmatrix} = \mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2 = \begin{bmatrix} w_{11}^2 x_1 + w_{12}^2 x_2 + b_1^2 \\ w_{21}^2 x_1 + w_{22}^2 x_2 + b_2^2 \end{bmatrix}$

Activation $\mathbf{a}^2 = \begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix} = \tanh(\mathbf{z}^2) = \begin{bmatrix} \tanh(w_{11}^2 x_1 + w_{12}^2 x_2 + b_1^2) \\ \tanh(w_{21}^2 x_1 + w_{22}^2 x_2 + b_2^2) \end{bmatrix}$

MLP: Matrix Notation (Layer 3)



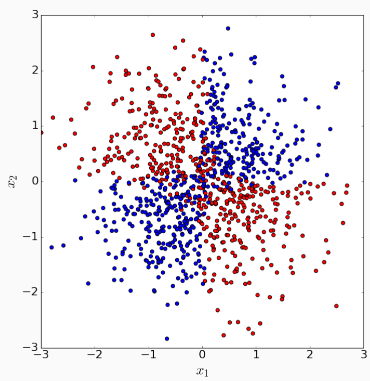
Weights are row vectors $\mathbf{W}^3 = \begin{bmatrix} w_{11}^3 & w_{12}^3 \end{bmatrix}$ Bias term vector $\mathbf{b}^3 = \begin{bmatrix} b_1^3 \end{bmatrix}$

Pre-activation $\mathbf{z}^3 = \begin{bmatrix} z_1^3 \end{bmatrix} = \mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3 = \begin{bmatrix} w_{11}^3 a_1^2 + w_{12}^3 a_2^2 + b_1^3 \end{bmatrix}$

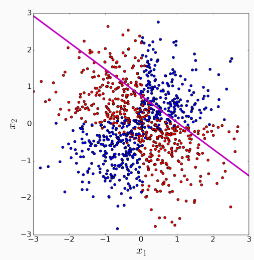
Activation $\mathbf{a}^3 = \begin{bmatrix} a_1^3 \end{bmatrix} = \sigma(\mathbf{z}^3) = \begin{bmatrix} \sigma(w_{11}^3 a_1^2 + w_{12}^3 a_2^2 + b_1^3) \end{bmatrix}$

MLP output: $\mathbf{y} = \mathbf{a}^3 = \sigma(\mathbf{z}^3)$

Classification Example: Logistic Regression vs Our MLP

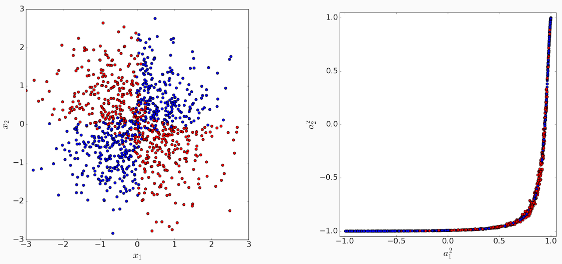


Logistic Regression Fails Badly



- No linear separator can separate the blue from the red
- Accuracy of logistic regression $\approx$ 50%
- What might work: Basis expansion, kernel methods
- We next consider our MLP example

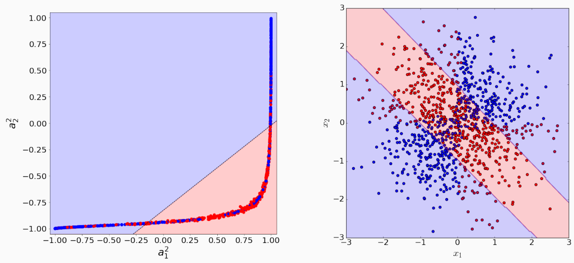
Scatterplot Comparison (x_1, x_2) vs (a_1^2, a_2^2)



Mapping input features x_1, x_2 to non-linear features a_1^2, a_2^2, where

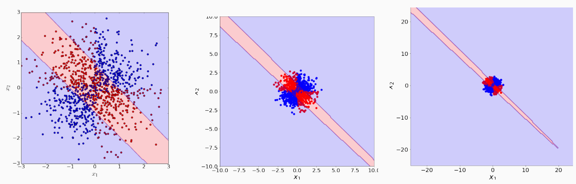
$$\mathbf{a}^2 = \begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix} = \tanh(\mathbf{W}^2\mathbf{x} + \mathbf{b}^2) = \begin{bmatrix} \tanh(w_{11}^2 x_1 + w_{12}^2 x_2 + b_1^2) \\ \tanh(w_{21}^2 x_1 + w_{22}^2 x_2 + b_2^2) \end{bmatrix}$$

MLP Decision Boundary in the (a_1^2, a_2^2) and (x_1, x_2) Planes



The linear separator in the (a_1^2, a_2^2) plane becomes non-linear in the (x_1, x_2) plane

MLP Decision Boundary: Zooming out of the (x_1, x_2) Plane



How To Learn the MLP Parameters?

- Compute the derivatives for all parameters of the MLP for each data point
- Average the derivatives over the training data points
  - Common: Average over a mini-batch instead of the entire dataset
- Perform a gradient descent step to update the model parameters
  - Learning the weights is guided by the error at every iteration
  - Error at output node gets back propagated to every layer
  - Weights adjusted in the next step proportionally to the error and weights
  - Larger weights get assigned larger error since they contribute more to the overall error

Background: Computing Derivatives

- Vector $\mathbf{z} \in \mathbb{R}^n$, function $f : \mathbb{R}^n \to \mathbb{R}$. Then $\frac{\partial f}{\partial \mathbf{z}} \in \mathbb{R}^n$ is the row vector

$$\frac{\partial f}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial f}{\partial z_1} & \cdots & \frac{\partial f}{\partial z_n} \end{bmatrix}$$

- Function $f : \mathbb{R}^n \to \mathbb{R}^m$, where $(f(\mathbf{z}))_i = f_i(\mathbf{z})$. Then $\frac{\partial f}{\partial \mathbf{z}}$ is the $m \times n$ Jacobian

$$\frac{\partial f}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & \frac{\partial f_1}{\partial z_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial z_1} & \cdots & \frac{\partial f_m}{\partial z_n} \end{bmatrix}$$

- Matrix $\mathbf{W} \in \mathbb{R}^{n \times m}$, function $f : \mathbb{R}^{n \times m} \to \mathbb{R}$. Then $\frac{\partial f}{\partial \mathbf{z}} \in \mathbb{R}^{n \times m}$ is the matrix

$$\frac{\partial f}{\partial \mathbf{W}} = \begin{bmatrix} \frac{\partial f}{\partial w_{11}} & \cdots & \frac{\partial f}{\partial w_{1m}} \\ \vdots & \vdots & \vdots \\ \frac{\partial f}{\partial w_{n1}} & \cdots & \frac{\partial f}{\partial w_{nm}} \end{bmatrix}$$

Background: Chain Rule for Multivariate Calculus

Functions $f : \mathbb{R}^n \to \mathbb{R}^k, g : \mathbb{R}^k \to \mathbb{R}^m, h = g \circ f : \mathbb{R}^n \to \mathbb{R}^m$

For $\mathbf{x} \in \mathbb{R}^n$, let $\mathbf{z} = f(\mathbf{x})$

$$\frac{\partial h}{\partial \mathbf{x}} = \frac{\partial h}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}}$$

- $\frac{\partial h}{\partial \mathbf{x}}$ is an $m \times n$ Jacobian matrix
- $\frac{\partial h}{\partial \mathbf{z}}$ is an $m \times k$ Jacobian matrix
- $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ is an $k \times n$ Jacobian matrix

Chain rule is the core technique behind the backpropagation algorithm

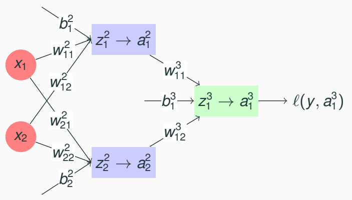
- Modular computation of derivatives of neural networks
- Efficient: Decompose derivatives into reusable building blocks

Alternative approach: Ad-hoc computation of derivatives for arbitrary functions

Computing the Derivatives for Our MLP

We can use the cross-entropy loss function in our example:

$$\ell(\mathbf{a}^3, y) = -(y\log(a_1^3) + (1-y)\log(1-a_1^3))$$



Want the derivatives

$\frac{\partial \ell}{\partial w_{11}^2}, \frac{\partial \ell}{\partial w_{12}^2}$

$\frac{\partial \ell}{\partial w_{21}^2}, \frac{\partial \ell}{\partial w_{22}^2}$

$\frac{\partial \ell}{\partial w_{11}^3}, \frac{\partial \ell}{\partial w_{12}^3}$

$\frac{\partial \ell}{\partial b_1^2}, \frac{\partial \ell}{\partial b_2^2}, \frac{\partial \ell}{\partial b_1^3}$

It suffices to compute $\frac{\partial \ell}{\partial \mathbf{z}^l}$, then derive $\frac{\partial \ell}{\partial w_{ij}^l}$ and $\frac{\partial \ell}{\partial b_i^l}$ from $\frac{\partial \ell}{\partial \mathbf{z}^l}$ using the chain rule

Working Out the Derivatives for Weights and Bias from Derivatives for $\mathbf{z}^l$

Assume: $\frac{\partial \ell}{\partial}$ and $\frac{\partial \ell}{\partial z^3}$ known. Want: $\frac{\partial \ell}{\partial w_{ij}^2}$; $i, j \in \{1,2\}$.

$$z_i^2 = \underline{w_{i1}^2 x_1 + w_{i2}^2 x_2} + \underline{b_i^2}$$

$$\frac{\partial \ell}{\partial w_{ij}^2} = \frac{\partial \ell}{\partial z_i^2} \underbrace{\frac{\partial z_i^2}{\partial w_{ij}^2}} = \frac{\partial \ell}{\partial z_i^2} x_j$$

$$\underbrace{\frac{\partial \ell}{\partial W^2}}_{\substack{2\times 2 \\ \text{Jacobian}}} = \begin{bmatrix} \frac{\partial \ell}{\partial w_{11}^2} & \frac{\partial \ell}{\partial w_{12}^2} \\ \frac{\partial \ell}{\partial w_{21}^2} & \frac{\partial \ell}{\partial w_{22}^2} \end{bmatrix} = \begin{bmatrix} \frac{\partial \ell}{\partial z_1^2} x_1 & \frac{\partial \ell}{\partial z_1^2} x_2 \\ \frac{\partial \ell}{\partial z_2^2} x_1 & \frac{\partial \ell}{\partial z_2^2} x_2 \end{bmatrix}.$$

$$//$$

Recall: $\frac{\partial \ell}{\partial z^2} = \begin{bmatrix} \frac{\partial \ell}{\partial z_1^2} & \frac{\partial \ell}{\partial z_2^2} \end{bmatrix}$

$$\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} \frac{\partial \ell}{\partial z_1^2} & \frac{\partial \ell}{\partial z_2^2} \end{bmatrix} \right)^T = \begin{bmatrix} \frac{\partial \ell}{\partial z_1^2} x_1 & \frac{\partial \ell}{\partial z_2^2} x_1 \\ \frac{\partial \ell}{\partial z_1^2} x_2 & \frac{\partial \ell}{\partial z_2^2} x_2 \end{bmatrix}^T$$

Working Out the Derivatives for Weights and Bias from Derivatives for $\mathbf{z}^l$

$$\boxed{\frac{\partial \ell}{\partial W^2} = \left(x \frac{\partial \ell}{\partial z^2}\right)^T}$$

Similarly, $\boxed{\frac{\partial \ell}{\partial W^3} = \left(a^2 \frac{\partial \ell}{\partial z^3}\right)^T}$

→ recall: $a^1 = x$

$$\frac{\partial \ell}{\partial W^2} = \left(\underline{a^1} \underline{\frac{\partial \ell}{\partial z^2}}\right)^T.$$

<u>Bias terms:</u> $\frac{\partial \ell}{\partial b_i^2} = \frac{\partial \ell}{\partial z_i^2} \cdot 1 \Rightarrow$

$$\boxed{\begin{aligned} \frac{\partial \ell}{\partial b^2} &= \frac{\partial \ell}{\partial z^2} \\ \frac{\partial \ell}{\partial b^3} &= \frac{\partial \ell}{\partial z^3} \end{aligned}}$$

Working Out the Derivatives for $\mathbf{z}^l$

Use cross-entropy loss function <u>in our example:</u>

$$\ell(a^3, y) = -\left[y \log(a_1^3) + (1-y)\log(1-a_1^3)\right]. \quad a^3 \in \mathbb{R}^1.$$

$$\frac{\partial \ell}{\partial a^3} = \left[\frac{\partial \ell}{\partial a_1^3}\right] = \left[-\left(\frac{y}{a_1^3} + \frac{1-y}{1-a_1^3}\cdot(-1)\right)\right] = \left[-\frac{y(1-a_1^3) - (1-y)a_1^3}{a_1^3(1-a_1^3)}\right]$$

$$= \left[-\frac{y - \cancel{y a_1^3} - a_1^3 + \cancel{y a_1^3}}{a_1^3 \cdot (1-a_1^3)}\right] = \left[\frac{a_1^3 - y}{a_1^3(1-a_1^3)}\right].$$

Our choice of activation function: $a_1^3 = \sigma(z_1^3)$.

$$\frac{\partial a^3}{\partial z^3} = \left[\sigma'(z_1^3)\right] = \left[a_1^3(1-a_1^3)\right].$$

then, $\boxed{\frac{\partial \ell}{\partial z^3}} = \frac{\partial \ell}{\partial a^3}\frac{\partial a^3}{\partial z^3} = \left[a_1^3 - y\right]$

Working Out the Derivatives for $\mathbf{z}^l$

Compute $\boxed{\frac{\partial \ell}{\partial z^2}}$. We will use the already computed $\frac{\partial \ell}{\partial z^3}$.

$$\underline{z^3 = W^3 a^2 + b^3.}$$

$$\frac{\partial z_i^3}{\partial a_j^2} = W_{ij}^3 \Rightarrow \boxed{\frac{\partial z^3}{\partial a^2} = W^3}$$

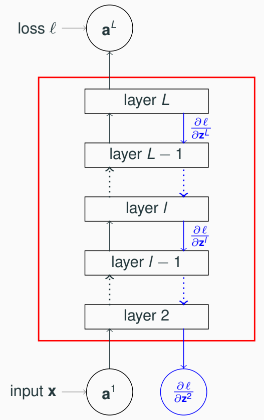
Recall: $a_i^2 = \tanh(z_i^2)$.

$$\boxed{\begin{aligned} \tanh(t) &= \frac{e^t - e^{-t}}{e^t + e^{-t}}. \\ \tanh'(t) &= 1 - (\tanh(t))^2. \end{aligned}}$$

$$\underbrace{\frac{\partial a^2}{\partial z^2}}_{\substack{2\times 2 \\ \text{Jacobian}}} = \begin{bmatrix} \frac{\partial a_1^2}{\partial z_1^2} & \frac{\partial a_1^2}{\partial z_2^2} \\ \frac{\partial a_2^2}{\partial z_1^2} & \frac{\partial a_2^2}{\partial z_2^2} \end{bmatrix} = \begin{bmatrix} (1-(a_1^2)^2) & 0 \\ 0 & (1-(a_2^2)^2) \end{bmatrix}.$$

$$\text{then, } \frac{\partial \ell}{\partial z^2} = \frac{\partial \ell}{\partial z^3}\frac{\partial z^3}{\partial a^2}\frac{\partial a^2}{\partial z^2} = \left[a_1^3 - y\right] W^3 \begin{bmatrix} 1-(a_1^2)^2 & 0 \\ 0 & 1-(a_2^2)^2 \end{bmatrix}.$$

The Backpropagation Algorithm



Consider all layers are fully connected: Every unit in layer $l-1$ has a connection with every unit in layer l

Each layer consists of a linear function and non-linear activation

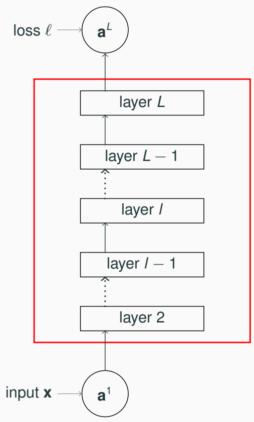
Layer l consists of the following:

$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$
$\mathbf{a}^l = f_l(\mathbf{z}^l)$

where f_l is the non-linear activation in layer l.

If there are n_l units in layer l, then $\mathbf{W}^l \in \mathbb{R}^{n_l \times n_{l-1}}$

Backward pass to compute derivatives



Forward Equations

(1) $\mathbf{a}^1 = \mathbf{x}$ (input)

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

(3) $\mathbf{a}^l = f_l(\mathbf{z}^l)$

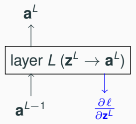
- f_l may map $\mathbf{z}^l$ to $\mathbf{a}^l$ elementwise
- f_l may map $\mathbf{z}^l$ to $\mathbf{a}^l$ vectorwise

$$\text{softmax}(\mathbf{z}) = \left[\frac{\exp(z_1)}{\sum_{j=1}^{n}\exp(z_j)}, \dots, \frac{\exp(z_n)}{\sum_{j=1}^{n}\exp(z_j)}\right]^\mathsf{T}$$

(4) $\ell(\mathbf{a}^L, y) = -\sum_{c=1}^{C} \mathbb{I}(y = c)\log(a_c^L)$

ℓ is the cross entropy loss function

Output Layer



$\mathbf{z}^L = \mathbf{W}^L \mathbf{a}^{L-1} + \mathbf{b}^L$

$\mathbf{a}^L = f_L(\mathbf{z}^L)$

Loss: $\ell(y, \mathbf{a}^L)$

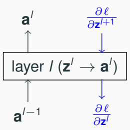
$\frac{\partial \ell}{\partial \mathbf{z}^L} = \frac{\partial \ell}{\partial \mathbf{a}^L}\frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L}$

If there are n_L (output) units in layer L, then $\frac{\partial \ell}{\partial \mathbf{a}^L}$ and $\frac{\partial \ell}{\partial \mathbf{z}^L}$ are row vectors with n_L elements and $\frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L}$ is the $n_L \times n_L$ Jacobian matrix:

$$\frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L} = \begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} & \frac{\partial a_1^L}{\partial z_2^L} & \cdots & \frac{\partial a_1^L}{\partial z_{n_L}^L} \\ \frac{\partial a_2^L}{\partial z_1^L} & \frac{\partial a_2^L}{\partial z_2^L} & \cdots & \frac{\partial a_2^L}{\partial z_{n_L}^L} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial a_{n_L}^L}{\partial z_1^L} & \frac{\partial a_{n_L}^L}{\partial z_2^L} & \cdots & \frac{\partial a_{n_L}^L}{\partial z_{n_L}^L} \end{bmatrix}$$

If f_L is applied element-wise, *e.g.*, sigmoid, then this matrix is diagonal

Back Propagation



$\mathbf{a}^l$ (the inputs into layer $l+1$)

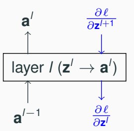
$\mathbf{z}^{l+1} = \mathbf{W}^{l+1}\mathbf{a}^l + \mathbf{b}^{l+1}$ ($w_{j,k}^{l+1}$ weight on connection from k^{th} unit in layer l to j^{th} unit in layer $l+1$)

$\mathbf{a}^l = f(\mathbf{z}^l)$ (f is a non-linearity)

$\frac{\partial \ell}{\partial \mathbf{z}^{l+1}}$ (derivative passed from layer above)

$$\begin{aligned} \frac{\partial \ell}{\partial \mathbf{z}^l} &= \frac{\partial \ell}{\partial \mathbf{z}^{l+1}}\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l} \\ &= \frac{\partial \ell}{\partial \mathbf{z}^{l+1}}\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{a}^l}\frac{\partial \mathbf{a}^l}{\partial \mathbf{z}^l} \\ &= \frac{\partial \ell}{\partial \mathbf{z}^{l+1}}\mathbf{W}^{l+1}\frac{\partial \mathbf{a}^l}{\partial \mathbf{z}^l} \end{aligned}$$

Derivatives with respect to Network Parameters



$\mathbf{z}^l = \mathbf{W}^l\mathbf{a}^{l-1} + \mathbf{b}^l$ ($w_{j,k}^l$ weight on connection from k^{th} unit in layer l-1 to j^{th} unit in layer l)

$\frac{\partial \ell}{\partial \mathbf{z}^l}$ (obtained using backpropagation)

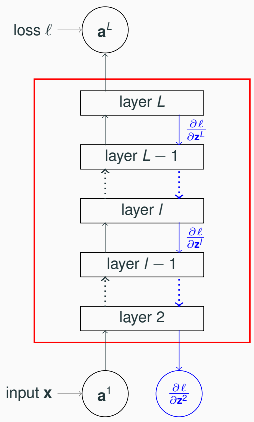
We obtain the derivatives wrt w_{ij}^l and b_i^l using $\frac{\partial \ell}{\partial \mathbf{z}^l}$:

- $\frac{\partial \ell}{\partial w_{ij}^l} = \frac{\partial \ell}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{ij}^l} = \frac{\partial \ell}{\partial z_i^l} a_j^{l-1}$
- $\frac{\partial \ell}{\partial b_i^l} = \frac{\partial \ell}{\partial z_i^l}$

More succinctly using matrix notation:

- $\frac{\partial \ell}{\partial \mathbf{W}^l} = \left(a^{l-1}\frac{\partial \ell}{\partial \mathbf{z}^l}\right)^\mathsf{T}$
- $\frac{\partial \ell}{\partial \mathbf{b}^l} = \frac{\partial \ell}{\partial \mathbf{z}^l}$

Backpropagation Algorithm: Forward and Backward Equations



Forward Equations

(1) $\mathbf{a}^1 = \mathbf{x}$ (input)

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

(3) $\mathbf{a}^l = f_l(\mathbf{z}^l)$

(4) $\ell(\mathbf{a}^L, y)$

Back-propagation Equations

(1) Compute $\frac{\partial \ell}{\partial \mathbf{z}^L} = \frac{\partial \ell}{\partial \mathbf{a}^L}\frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L}$

(2) $\frac{\partial \ell}{\partial \mathbf{z}^l} = \frac{\partial \ell}{\partial \mathbf{z}^{l+1}}\mathbf{W}^{l+1}\frac{\partial \mathbf{a}^l}{\partial \mathbf{z}^l}$

(3) $\frac{\partial \ell}{\partial \mathbf{W}^l} = \left(\mathbf{a}^{l-1}\frac{\partial \ell}{\partial \mathbf{z}^l}\right)^\mathsf{T}$

(4) $\frac{\partial \ell}{\partial \mathbf{b}^l} = \frac{\partial \ell}{\partial \mathbf{z}^l}$

Computational Questions

What is the running time to compute the gradient for a single data point?

- As many matrix multiplications as there are fully connected layers
- Performed twice during forward and backward pass

What is the space requirement?

- Need to store vectors $\mathbf{a}^l$, $\mathbf{z}^l$, and $\frac{\partial \ell}{\partial \mathbf{z}^l}$ for each layer

Can we process multiple examples together?

- Yes, if we minibatch, we perform tensor operations
- Make sure that all parameters fit in GPU memory