How the backpropagation algorithm works

Srikumar Ramalingam

School of Computing

University of Utah

Reference

Most of the slides are taken from the second chapter of the online book by Michael Nielson:

- neuralnetworksanddeeplearning.com

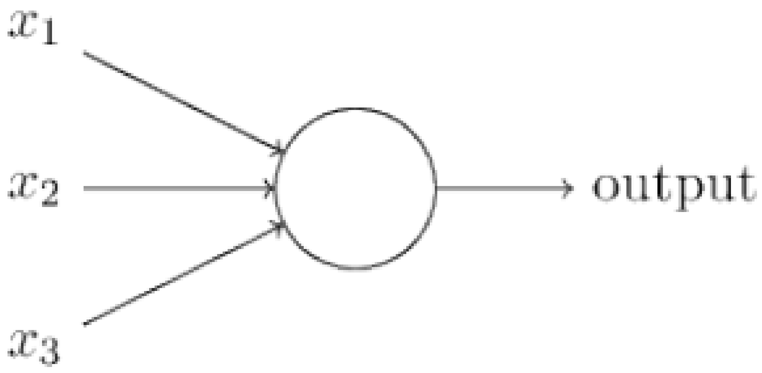
Introduction

- First discovered in 1970.
- First influential paper in 1986:

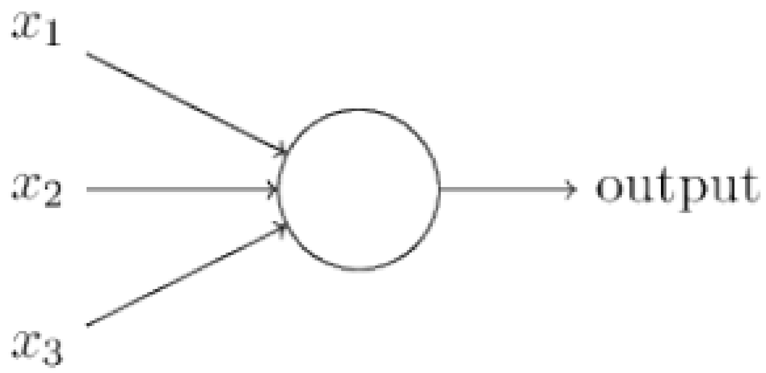
Rumelhart, Hinton and Williams, Learning representations by back-propagating errors, Nature, 1986.

Perceptron (Reminder)

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



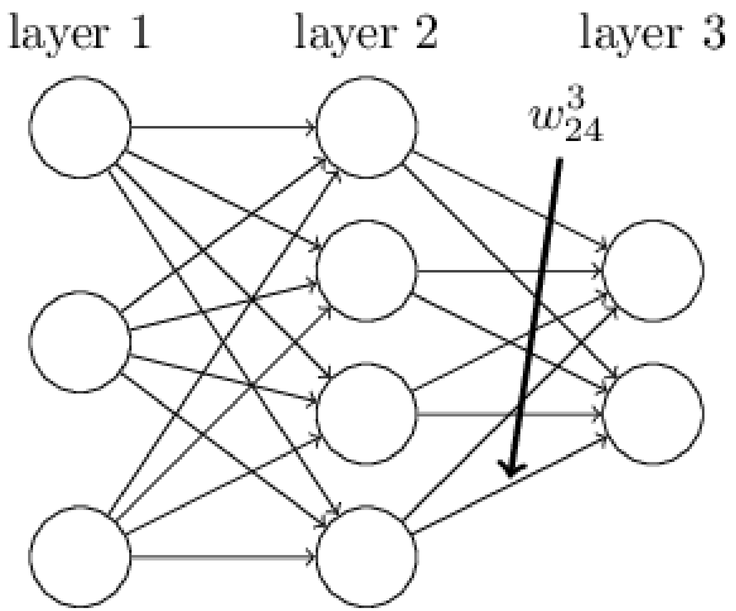
Sigmoid neuron (Reminder)



- A sigmoid neuron can take real numbers (x_1, x_2, x_3) within 0 to 1 and returns a number within 0 to 1. The weights (w_1, w_2, w_3) and the bias term b are real numbers.

Sigmoid function $$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$

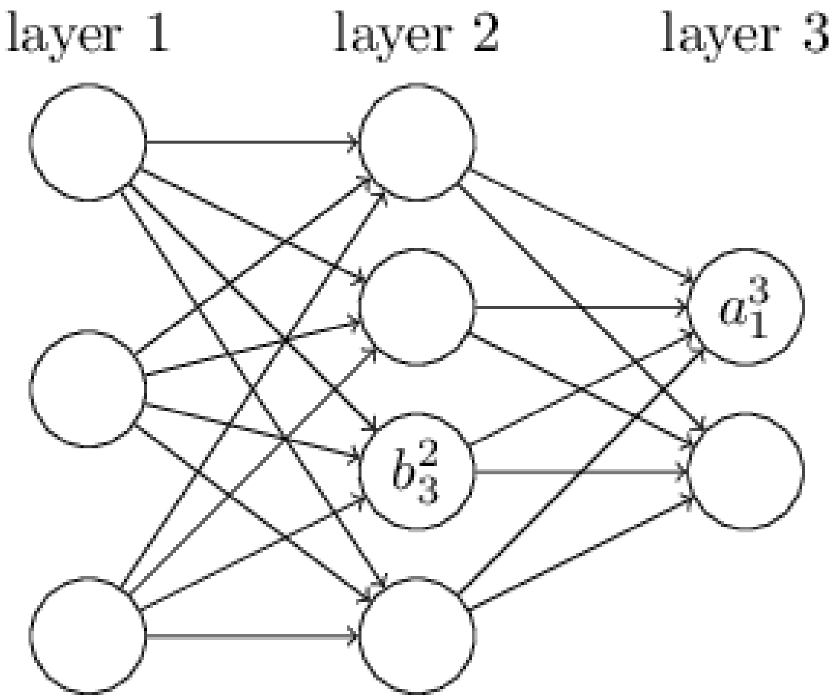
Matrix equations for neural networks



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

The indices “j” and “k” seem a little counter-intuitive!

Layer to layer relationship



$$a_j^l = \sigma(z_j^l)$$

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

- b_j^l is the bias term in the jth neuron in the lth layer.
- a_j^l is the activation in the jth neuron in the lth layer.
- z_j^l is the weighted input to the jth neuron in the lth layer.

Cost function from the network

Groundtruth for each input

Output activation vector for a specific training sample x.

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

# of input samples

for each input sample

Backpropagation and stochastic gradient descent

- The goal of the backpropagation algorithm is to compute the gradients $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ of the cost function C with respect to each and every weight and bias parameters. Note that backpropagation is only used to compute the gradients.

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

- Stochastic gradient descent is the training algorithm.

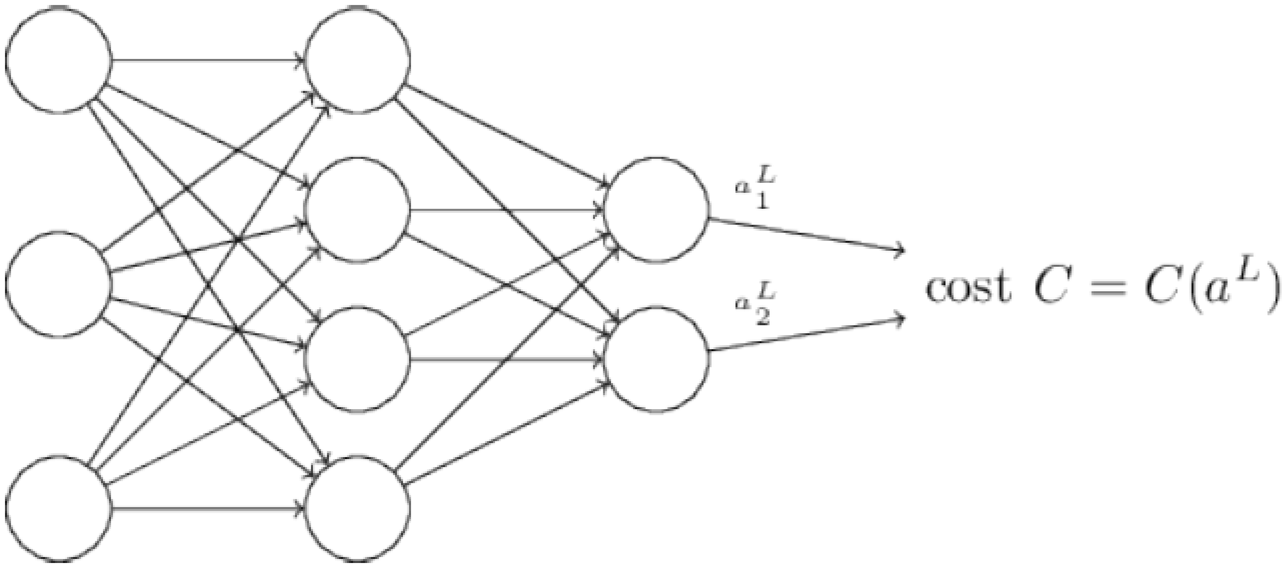
Assumptions on the cost function

1. We assume that the cost function can be written as the average over the cost functions from individual training samples: $C = \frac{1}{n}\sum_x C_x$. The cost function for the individual training sample is given by $C_x = \frac{1}{2}|y(x) - a^L(x)|^2$.

- why do we need this assumption? Backpropagation will only allow us to compute the gradients with respect to a single training sample as given by $\frac{\partial C_x}{\partial w}$ and $\frac{\partial C_x}{\partial b}$. We then recover $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ by averaging the gradients from the different training samples.

Assumptions on the cost function (continued)

2. We assume that the cost function can be written as a function of the output from the neural network. We assume that the input x and its associated correct labeling $y(x)$ are fixed and treated as constants.



Hadamard product

- Let s and t are two vectors. The Hadamard product is given by:

$$s \odot t$$

$$(s \odot t)_j = s_j t_j$$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

Such elementwise multiplication is also referred to as schur product.

Backpropagation

- Our goal is to compute the partial derivatives $\frac{\partial C}{\partial w_{jk}^{l}}$ and $\frac{\partial C}{\partial b_{j}^{l}}$.
- We compute some intermediate quantities while doing so:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

Four equations of the BP (backpropagation)

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \qquad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \qquad \text{(BP2)}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \qquad \text{(BP3)}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \qquad \text{(BP4)}$$

Chain Rule in differentiation

- In order to differentiate a function $\mathrm{z} = f\left(g(x)\right)$ w.r.t x, we can do the following:

Let $\mathrm{y} = g(x), \;\; z = f(y), \; \frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx}$

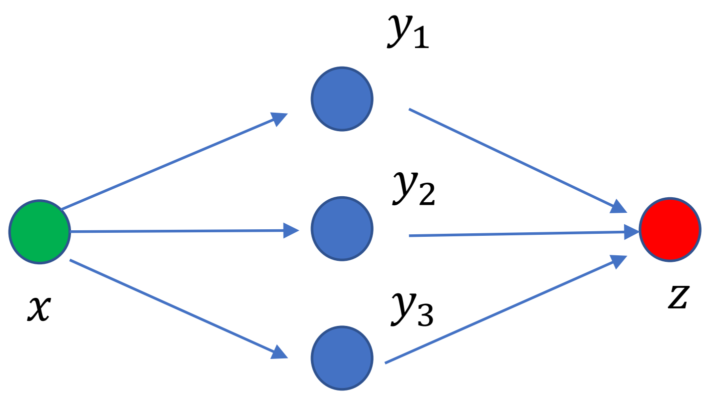
Chain Rule in differentiation (vector case)

Let $x \in \mathbb{R}^m, y \in \mathbb{R}^n$, g maps from $\mathbb{R}^m$ to $\mathbb{R}^n$, and f maps from $\mathbb{R}^n$ to $\mathbb{R}$. If $y = g(x)$ and $z = f(y)$, then

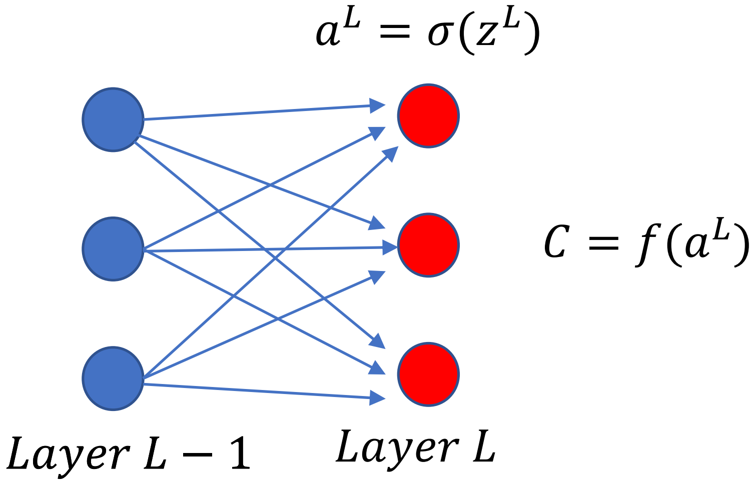
$$\frac{\partial z}{\partial x_i} = \sum_k \frac{\partial z}{\partial y_k} \frac{\partial y_k}{\partial x_i}$$

Chain Rule in differentiation (computation graph)

$$\frac{\partial z}{\partial x} = \sum_{\substack{j:x\in Parent(y_j),\\ y_j\in Ancestor\,(z)}} \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x}$$



BP1



$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

Here L is the last layer.

$$\delta^L = \frac{\partial C}{\partial z^L}, \qquad \sigma'(z^L) = \frac{\partial\left(\sigma(z^L)\right)}{\partial z^L}, \qquad \nabla_a C = \frac{\partial C}{\partial a^L} = \left(\frac{\partial C}{\partial a_1^L}, \frac{\partial C}{\partial a_2^L}, \dots, \frac{\partial C}{\partial a_n^L}\right)^T$$

Proof:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}$$ when $j \neq k$, the term $\frac{\partial a_k^L}{\partial z_j^L}$ vanishes.

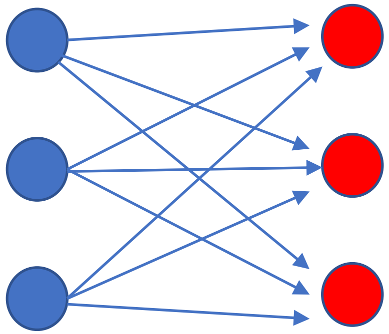
$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

Thus we have

$$\partial^L = \frac{\partial C}{\partial a^L} \odot \sigma'(z^L)$$

$$z^{l+1} = w^{l+1}a^{l} + b^{l+1}$$

BP2



$Layer\ l$ $\quad$ $Layer\ l+1$

$$\partial^{l} = \left(\left(w^{l+1}\right)^{T}\delta^{l+1}\right) \odot \sigma'(z^{l})$$

Proof:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}$$

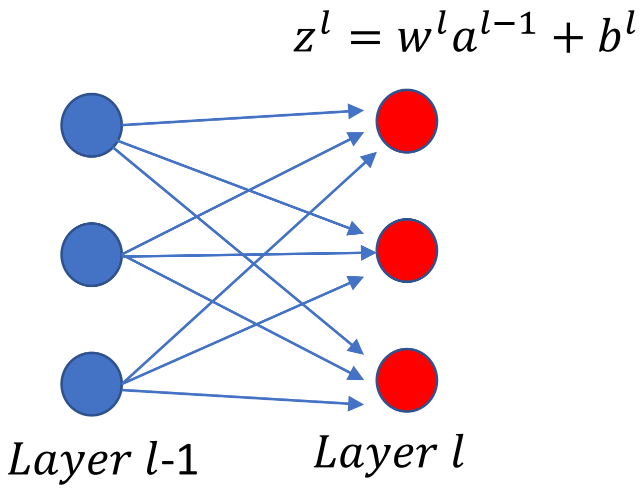
$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^l = \sum_j w_{kj}^{l+1} \sigma\left(z_j^l\right) + b_k^l$$

By differentiating we have:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'\left(z_j^l\right)$$

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'\left(z_j^l\right)$$

BP3



$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

Proof:

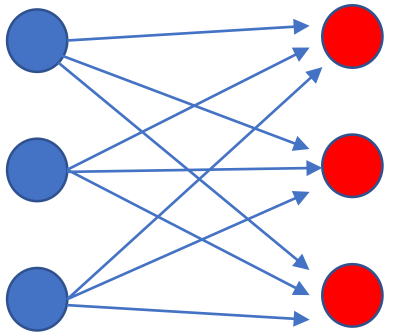
$$\frac{\partial C}{\partial b_j^l} = \sum_k \left(\frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_j^l} \right) = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l}$$

$$= \delta_j^l \frac{\partial \left(\sum_k \left(w_{jk} a_k^{l-1} + b_j^l \right) \right)}{\partial b_j}$$

$$= \delta_j^l$$

BP4

$z^l = w^l a^{l-1} + b^l$



$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

Proof:

$$\frac{\partial C}{\partial w^l_{jk}} = \sum_m \frac{\partial C}{\partial z^l_m} \frac{\partial z^l_m}{\partial w^l_{jk}}$$

$$= \frac{\partial C}{\partial z^l_j} \frac{\partial z^l_j}{\partial w_{jk}}$$

$$= \delta^l_j \frac{\partial\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)}{\partial w_{jk}}$$

$$= \delta^l_j \; a^{l-1}_k$$

The backpropagation algorithm

1. **Input x:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \dots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

3. **Output error δ^L:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \dots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

The word "backpropagation" comes from the fact that we compute the error vectors δ_j^l in the backward direction.

Stochastic gradient descent with BP

1. **Input a set of training examples**
2. **For each training example x:** Set the corresponding input activation $a^{x,1}$, and perform the following steps:
   - **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.
   - **Output error $\delta^{x,L}$:** Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
   - **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.
3. **Gradient descent:** For each $l = L, L-1, \ldots, 2$ update the weights according to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

Gradients using finite differences

$$\frac{\partial C}{\partial w_j} \approx \frac{C(w + \epsilon e_j) - C(w)}{\epsilon}$$

Here ϵ is a small positive number and e_j is the unit vector in the jth direction.
Conceptually very easy to implement.
In order to compute this derivative w.r.t one parameter, we need to do one forward pass – for millions of variables we will have to do millions of forward passes.

- Backpropagation can get all the gradients in just one forward and backward pass – forward and backward passes are roughly equivalent in computations.

The derivatives using finite differences would be a million times slower!!

Backpropagation – the big picture

$$\Delta C \approx \sum_{mnp\ldots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

- To compute the total change in C we need to consider all possible paths from the weight to the cost

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_{mnp\ldots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l}$$

- We are computing the rate of change of C w.r.t a weight w.
- Every edge between two neurons in the network is associated with a rate factor that is just the ratio of partial derivatives of one neurons activation with respect to another neurons activation.
- The rate factor for a path is just the product of the rate factors of the edges in the path.
- The total change is the sum of the rate factors of all the paths from the weight to the cost.

Thank You

Chain Rule in differentiation (vector case)

Let $x \in \mathbb{R}^m, y \in \mathbb{R}^n$, g maps from $\mathbb{R}^m$ to $\mathbb{R}^n$, and f maps from $\mathbb{R}^n$ to $\mathbb{R}$. If $y = g(x)$ and $z = f(y)$, then

$$\frac{\partial z}{\partial x_i} = \sum_k \frac{\partial z}{\partial y_k} \frac{\partial y_k}{\partial x_i}$$

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z$$

Here $\left(\frac{\partial y}{\partial x}\right)$ is the $n \times m$ Jacobian matrix of g.

DERIVATIVE RULES

$$\frac{d}{dx}\left(x^n\right) = nx^{n-1}$$

$$\frac{d}{dx}\left(\sin x\right) = \cos x$$

$$\frac{d}{dx}\left(\cos x\right) = -\sin x$$

$$\frac{d}{dx}\left(a^x\right) = \ln a \cdot a^x$$

$$\frac{d}{dx}\left(\tan x\right) = \sec^2 x$$

$$\frac{d}{dx}\left(\cot x\right) = -\csc^2 x$$

$$\frac{d}{dx}\left(f(x)\cdot g(x)\right) = f(x)\cdot g'(x) + g(x)\cdot f'(x)$$

$$\frac{d}{dx}\left(\sec x\right) = \sec x \tan x$$

$$\frac{d}{dx}\left(\csc x\right) = -\csc x \cot x$$

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{g(x)\cdot f'(x) - f(x)\cdot g'(x)}{\left(g(x)\right)^2}$$

$$\frac{d}{dx}\left(\arcsin x\right) = \frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\left(\arctan x\right) = \frac{1}{1+x^2}$$

$$\frac{d}{dx}\left(f(g(x))\right) = f'(g(x))\cdot g'(x)$$

$$\frac{d}{dx}\left(\operatorname{arc\,sec} x\right) = \frac{1}{x\sqrt{x^2-1}}$$

$$\frac{d}{dx}\left(\ln x\right) = \frac{1}{x}$$

$$\frac{d}{dx}\left(\sinh x\right) = \cosh x$$

$$\frac{d}{dx}\left(\cosh x\right) = \sinh x$$

Source: http://math.arizona.edu/~calc/Rules.pdf

INTEGRAL RULES

$$\int x^n dx = \frac{1}{n+1} x^{n+1} + c, \quad n \neq -1$$

$$\int a^x dx = \frac{1}{\ln a} a^x + c$$

$$\int \frac{1}{x} dx = \ln|x| + c$$

$$\int \frac{dx}{\sqrt{1-x^2}} = \arcsin x + c$$

$$\int \frac{dx}{1+x^2} = \arctan x + c$$

$$\int \frac{dx}{x\sqrt{x^2-1}} = \operatorname{arc\,sec} x + c$$

$$\int \sin x dx = -\cos x + c$$

$$\int \cos x dx = \sin x + c$$

$$\int \sec^2 x dx = \tan x + c$$

$$\int \sinh x dx = \cosh x + c$$

$$\int \csc^2 x dx = -\cot x + c$$

$$\int \sec x \tan x dx = \sec x + c$$

$$\int \csc x \cot x dx = -\csc x + c$$

$$\int \cosh x dx = \sinh x + c$$

Source: http://math.arizona.edu/~calc/Rules.pdf