Math Primer and Neural Network Basics

Deep Learning and AI, WS21/22

Dr. Yinchong Yang

Siemens AG

27.10.2021

Table of Contents

Table of Contents

Probability Theory

Maximum Likelihood Estimation

- Assumption : observations $\boldsymbol{x} = \{x_i\}_{i=1}^n, n \geq 1$ follow a distribution with a PDF $p_X(x|\theta)$ parameterized by θ
- The likelihood function :

$$\mathcal{L}(\theta|\boldsymbol{x}) = \prod_{i=1}^{n} p_X(x_i|\theta),$$

  - is a function of the parameter θ as variable conditioned on observed data points, and
  - quantifies the plausibility of θ given specific observed data points $\boldsymbol{x}$.

- It is often more comfortable to work with the *log likelihood* function

$$\ell(\theta|\boldsymbol{x}) = \sum_{i=1}^{n} \log p_X(x_i|\theta) \tag{1}$$

because one has summation instead of product, and log likelihood is under certain circumstances concave in θ ("log concavity").

Probability Theory

Maximum Likelihood Estimation

- We are interested in finding a distribution $p_X(\cdot|\theta)$ that is most likely to have generated the observations $\boldsymbol{x}$, i.e.

$$\hat{\theta}_{ML} = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\theta|\boldsymbol{x}) = \underset{\theta}{\operatorname{argmax}}\, \ell(\theta|\boldsymbol{x})$$

- MLE is often realized by finding the θ that would set the first derivative to 0.
- Example 1 : $p_X(x|\theta) = \mathcal{N}(x|\mu, \Sigma)$

$$\ell(\mu, \sigma|\boldsymbol{x}) \overset{(1)}{=} \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{(x_i - \mu)^2}{2\sigma^2}$$

$$\frac{\partial}{\partial \mu}\ell(\mu, \sigma|\boldsymbol{x}) = \sum_{i=1}^{n} \frac{x_i - \mu}{\sigma^2} \overset{!}{=} 0 \Rightarrow \hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\frac{\partial}{\partial \sigma}\ell(\mu, \sigma|\boldsymbol{x}) = \sum_{i=1}^{n} -\sigma^{-1} + \frac{(x_i - \mu)^2}{\sigma^3} \overset{!}{=} 0 \Rightarrow \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$$

Probability Theory

Maximum Likelihood Estimation

- Example 2 : $p_X(x|\theta) = \mathcal{B}(x|p, 1)$

$$\ell(p|\boldsymbol{x}) = \sum_{i=1}^{n} x_i \log(p) + (1 - x_i) \log(1 - p))$$

$$\frac{\partial}{\partial p}\ell(p|\boldsymbol{x}) = \frac{\sum_{i=1}^{n} x_i}{p} - \frac{n - \sum_{i=1}^{n} x_i}{1 - p} \stackrel{!}{=} 0 \Rightarrow \hat{p} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\text{with } \frac{\partial^2}{\partial^2 p}\ell(p|\boldsymbol{x}) = \underbrace{-\frac{\sum_{i=1}^{n} x_i}{p^2}}_{\leq 0} \underbrace{-\frac{n - \sum_{i=1}^{n} x_i}{(1-p)^2}}_{\leq 0} \leq 0 \Rightarrow \text{concavity}$$

Probability Theory

Maximum Likelihood Estimation

- MLE as a classical framework in statistical inference.
- But in supervised learning, we have *i.i.d.* observation *pairs* $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$.
- The likelihood becomes

$$\ell(\theta|\boldsymbol{X}, \boldsymbol{y}) = \sum_{i=1}^{n} \log p_Y(y_i|\theta_i), \text{ with } \theta_i = \theta(\boldsymbol{x}_i)$$

by having θ depend on $\boldsymbol{x}_i$, e.g.

$$\theta(\boldsymbol{x}_i) = \eta(\boldsymbol{x}_i|\boldsymbol{\beta}).$$

- $\ell(\theta|\boldsymbol{X}, \boldsymbol{y})$ quantifies the plausibility of observing y_i given $\boldsymbol{x}_i$ and β, where
- $p_Y(y|\theta)$ encodes our distribution assumption about y, and
- $\eta(\boldsymbol{x}|\boldsymbol{\beta})$ represents our model.

Probability Theory

Maximum Likelihood Estimation

- In statistical inference, we are interested in finding one θ that best explains all $\{x_i\}_{i=1}^n$.
- In supervised learning, we attempt to find a θ_i for each y_i to look plausible.
- The same principle of MLE still applies, i.e. we maximize the likelihood

$$\sum_{i=1}^{n} \log p_Y(y_i|\theta_i) = \sum_{i=1}^{n} \log p_Y(y_i|\eta(\boldsymbol{x}_i|\boldsymbol{\beta}))$$

  via optimizing the only variable in the system : $\boldsymbol{\beta}$.
- Only in rare cases can we have a closed form solution for $\boldsymbol{\beta}$ since it often enters the system in a non-linear way.
- Numerical optimization serves as standard toolbox.

Probability Theory

Maximum a Posteriori Estimation

- MLE attempts to find the θ that makes $\boldsymbol{x}$ most plausible in the sense that

$$\mathcal{L}(\theta|\boldsymbol{x}) \to \max.$$

- A Bayesian paradigm to find the most plausible θ with
  - A prior distribution assumption $p(\theta|\alpha)$,
  - A means to update the prior distribution with data $\boldsymbol{x}$.

$$\begin{aligned} p(\theta|\boldsymbol{x}) = \frac{p(\boldsymbol{x},\theta)}{p(\boldsymbol{x})} &\propto p(\boldsymbol{x}|\theta)p(\theta) = \prod_{i=1}^{n} p_X(x_i|\theta)p(\theta|\alpha) \\ &= \mathcal{L}(\theta|\boldsymbol{x})p(\theta|\alpha) \end{aligned}$$

$$\hat{\theta}_{MAP} = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\theta|\boldsymbol{x})p(\theta|\alpha)$$

Probability Theory

Maximum a Posteriori Estimation

- Example 1 [1] : statistical inference with prior $p(\theta|\alpha) = \mathcal{N}(\mu|\mu_0, \sigma_0^2)$

$$
\begin{aligned}
p(\mu|\boldsymbol{x}) &\propto \mathcal{L}(\mu|\boldsymbol{x})p(\mu|\alpha) \\
&= \prod_{i=1}^{n} \mathcal{N}(x_i|\mu, \sigma^2)\mathcal{N}(\mu|\mu_0, \sigma_0^2) \\
&= \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left(-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}\right) \frac{1}{\sqrt{2\pi}\sigma_0} \exp\left(-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}\right) \\
&= \ldots \text{ ("Completing the square")} \\
&= \mathcal{N}(\mu|\mu_{AP}, \sigma_{AP}^2)
\end{aligned}
$$

$$
\mu_{AP} = \frac{\sigma^2}{n\sigma_0^2+\sigma^2}\mu_0 + \frac{n\sigma_0^2}{n\sigma_0^2+\sigma^2}\hat{\mu}_{ML}, \quad \sigma_{AP}^2 = \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right)^{-1}
$$

- for $n \to \infty$, $\mu_{AP} \to \hat{\mu}_{ML}$ and $\sigma_{AP}^2 \to 0$: dominated by data.
- for $n = 0$, $\mu_{AP} = \mu_0$ and $\sigma_{AP}^2 = \sigma_0^2$: dominated by prior.

1. Bishop 2006, section 2.3.6, p97-98

Probability Theory

Maximum a Posteriori Estimation

- Example 2 : (Pseudo-)Bayesian linear regression $f_Y(y_i|\theta_i) = \mathcal{N}(\boldsymbol{x}_i^T\boldsymbol{\beta}, \sigma^2)$ with prior $p(\theta|\alpha) = \mathcal{N}(\boldsymbol{\beta}|\mathbf{0}, \sigma_0^2\boldsymbol{I})$

$$
\begin{aligned}
p(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y}) &\propto \mathcal{L}(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y})p(\boldsymbol{\beta}|\alpha) \\
\log p(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y}) &\propto \ell(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y}) + \log p(\boldsymbol{\beta}|\alpha) \\
&= \sum_{i=1}^{n} \log p_Y(y_i|\eta(\boldsymbol{x}_i|\boldsymbol{\beta})) + \log \mathcal{N}(\boldsymbol{\beta}|\mathbf{0}, \sigma_0^2\boldsymbol{I}) \\
&= \sum_{i=1}^{n} \log p_Y(y_i|\eta(\boldsymbol{x}_i|\boldsymbol{\beta})) - \frac{1}{2\sigma_0^2}\boldsymbol{\beta}^T\boldsymbol{\beta} + c
\end{aligned}
$$

  The Gaussian prior provides a $l2$-norm regularization on the model parameters, while the Laplace prior gives $l1$ regularization.

- *Fully* Bayesian often implies uncertainty of the prediction [2] :

$$
\begin{aligned}
p(y_*|\boldsymbol{X}, \boldsymbol{y}) &= \int p(y_*|\boldsymbol{\beta})p(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y})d\boldsymbol{\beta} \\
&= \mathcal{N}(y_*|\sigma^{-2}\boldsymbol{x}_*^T\boldsymbol{S}^{-1}\boldsymbol{X}^T\boldsymbol{y}, \sigma^2 + \boldsymbol{x}_*^T\boldsymbol{S}^{-1}\boldsymbol{x}_*), \ \boldsymbol{S} = \sigma^{-2}\boldsymbol{X}^T\boldsymbol{X} + \sigma_0^{-2}\boldsymbol{I}
\end{aligned}
$$

2. Bishop 2006, section 3.3.2, p156

~~Numerical~~ Analysis

Vector and matrix calculus

- A scalar function $J \in \mathbb{R}$ of vector $\boldsymbol{\theta} \in \mathbb{R}^p$, the 1st-order derivative (gradient) :

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \left[\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} \; \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} \; \cdots \; \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_p} \right]^T = \left(\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T} \right)^T \tag{2}$$

- A vector function $\boldsymbol{J} \in \mathbb{R}^k$ of vector $\boldsymbol{\theta} \in \mathbb{R}^p$, the 1st-order derivative (Jacobian)

$$\frac{\partial \boldsymbol{J}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T} \overset{(i)}{=} \begin{bmatrix} \frac{\partial \boldsymbol{J_1}(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial \boldsymbol{J_1}(\boldsymbol{\theta})}{\partial \theta_2} & \cdots & \frac{\partial \boldsymbol{J_1}(\boldsymbol{\theta})}{\partial \theta_p} \\ \frac{\partial \boldsymbol{J_2}(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial \boldsymbol{J_2}(\boldsymbol{\theta})}{\partial \theta_2} & \cdots & \frac{\partial \boldsymbol{J_2}(\boldsymbol{\theta})}{\partial \theta_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \boldsymbol{J}_k(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial \boldsymbol{J}_k(\boldsymbol{\theta})}{\partial \theta_2} & \cdots & \frac{\partial \boldsymbol{J}_k(\boldsymbol{\theta})}{\partial \theta_p} \end{bmatrix} \overset{(ii)}{=} \left(\frac{\partial \boldsymbol{J}(\boldsymbol{\theta})^T}{\partial \boldsymbol{\theta}} \right)^T \tag{3}$$

(i) by applying Eq. (2) from right to left, and (ii) from left to right.

~~Numerical~~ Analysis

Vector and matrix calculus

- A scalar function $J \in \mathbb{R}$ of $\boldsymbol{\theta} \in \mathbb{R}^p$, 2nd order derivative (Hessian) :

$$\begin{aligned}
\frac{\partial^2 J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} &= \frac{\partial}{\partial \boldsymbol{\theta}} \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T} \\
&= \frac{\partial}{\partial \boldsymbol{\theta}} \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} & \cdots & \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_p} \end{bmatrix} && \text{| Eq. (2)} \\
&= \begin{bmatrix} \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_1} & \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_p} \\ \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_2} & \cdots & \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_p \partial \theta_1} & \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_p \partial \theta_2} & \cdots & \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \theta_p \partial \theta_p} \end{bmatrix} && \text{| Eq. (3)}
\end{aligned}$$

Numerical Analysis

Optimization algorithms

An optimization problem :

$$\arg\min_{\theta} J(\theta) \text{ where } \theta \in \mathbb{R}^p \text{ and } J : \mathbb{R}^p \to \mathbb{R} \tag{4}$$

Subtypes :

- unconstrained v.s. constrained problem
  Eq. 4 becomes constrained if

$$\arg\min_{\theta} J(\theta) \text{ subject to } c_i(\theta) = 0,\ i \in \mathcal{I} \text{ and } c_k(\theta) \geq 0,\ k \in \mathcal{K} \tag{5}$$

- Linear v.s. non-linear problems
  Eq. 5 is nonlinear if J is nonlinear in θ or any c_i, c_k is nonlinear in θ.

Numerical Analysis

Optimization algorithms

Subtypes :

- Convex and non-convex problems

$$J(\alpha\theta_1 + (1-\alpha)\theta_2) \leq \alpha J(\theta_1) + (1-\alpha)J(\theta_2) \ \forall \ \alpha \in [0,1]$$

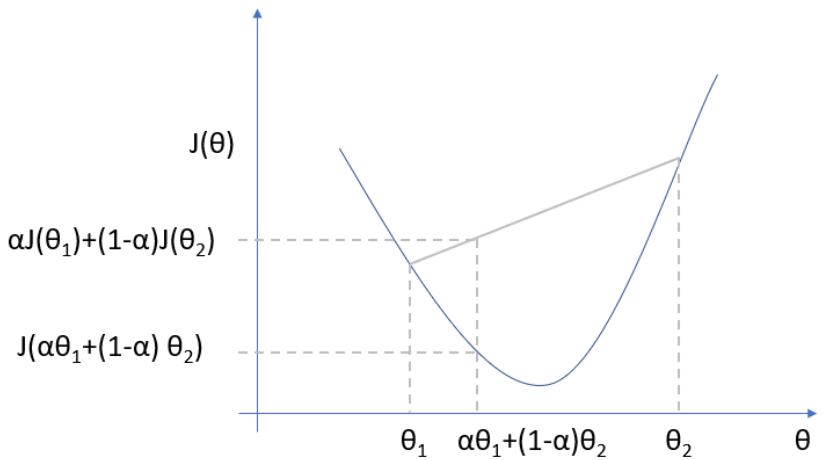


Figure – Illustration of the definition of a function being convex

- A local minimum is also a global minimum of a convex function.
- A univariate function is convex if the second derivative is non-negative everywhere.
- A function is convex if the Hessian is positive semidefinite.

Numerical Analysis

Optimization algorithms

2nd order algorithm : Newton-Raphson method

- Find the root for a function $f(\theta)$, i.e. $f(\theta_*) = 0$:

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

- Minimizing $J(\theta)$ equivalent to finding the root for $\frac{\partial J(\theta)}{\partial \theta}$ as long as $J(\theta)$ is convex, i.e. $\frac{\partial^2 J(\theta)}{\partial \theta \partial \theta^T}$ is positive semidefinite ;
- General update rule :

$$\theta := \theta - \left(\frac{\partial J(\theta)}{\partial \theta \partial \theta^T} \right)^{-1} \frac{\partial J(\theta)}{\partial \theta}$$

- In ML applicable for smaller models where one can write down the Hessian which is of reasonable size ;

Numerical Analysis

Optimization algorithms

1st order algorithms : Gradient descent

- In case the Hessian matrix (and inverse !) is too expensive or impossible to calculate : Quasi-Newton with approximated Hessian.
- The approximated Hessian could be still too large, especially for deep neural networks.
- Update rule of gradient descent :

$$\theta := \theta - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta} \tag{6}$$

- α as one of the most important hyper parameter in training.
- Actually, the optimal choice of α could be $\frac{1}{\lambda}$ with λ being the maximal eigenvalue of the Hessian. Recall $\boldsymbol{Ax} = \lambda \boldsymbol{x}$.
- In ML : the dominating algorithm to train deep neural networks (Next week with details).

Numerical Analysis

Optimization algorithms

Background : Why does gradient descent work ? [3]
Recall Taylor approximation : given a function f and a value a, the function evaluated at value x is approximately

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots$$

First order approximation of the objective function at a new value $\boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

$$T_1 J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \left(\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right)^T \cdot \Delta\boldsymbol{\theta}$$

Embed the gradient update rule Eq. (6) : $\boldsymbol{\theta} := \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$ with $\Delta\boldsymbol{\theta} = -\alpha \cdot \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$

$$\begin{aligned} J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + \left(\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right)^T \cdot (-\alpha \cdot \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}) \\ &= J(\boldsymbol{\theta}) - \alpha \cdot \left(\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right)^T \left(\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right) \leq J(\boldsymbol{\theta}) \end{aligned}$$

Conclusion : in the close neighborhood of θ, where the first order Taylor approximation applies, moving in the gradient direction will not increase the objective.

3. K. Weinberger, CS4780 Cornell

Numerical Analysis

Optimization algorithms

2nd order Taylor approximation of the objective function :

$$T_2 J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot \Delta\boldsymbol{\theta} + \frac{1}{2} \Delta\boldsymbol{\theta}^T \frac{\partial^2 J(\boldsymbol{\theta})}{\partial^2 \boldsymbol{\theta}} \Delta\boldsymbol{\theta}$$

One finds the optimal $\Delta\theta$ by setting the 1st derivative to be 0 and solve the equation w.r.t. $\Delta\boldsymbol{\theta}$

$$\frac{\partial T_2 J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})}{\partial \Delta\theta} = 0 + \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \frac{\partial^2 J(\boldsymbol{\theta})}{\partial^2 \boldsymbol{\theta}} \cdot \Delta\boldsymbol{\theta} := 0$$

$$\Delta\theta = -\left(\frac{\partial^2 J(\boldsymbol{\theta})}{\partial^2 \boldsymbol{\theta}} \right)^{-1} \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

$\rightarrow$ The Newton-Raphson update rule.

Numerical Analysis

Optimization algorithms

0-order algorithms :

- Not even the first order derivative is known : black box models.
- At least the evaluation of the model at various parameter configurations $J(\theta)$ should be doable.
- $J(\theta)$ does not need to be differentiable : any metrics such as AUROC, f1-score, rankings...
- Evolution strategy in the simplest form (single winner) :

$$\theta := \arg\min_i J(\theta + \epsilon_i) \text{ with } \epsilon_i \sim N(0, \sigma^2)$$

- Hyper-parameter : number of noises and winners.
- Nelder Mead, Bayesian optimization.
- In ML : reinforcement learning, parallel and distributed training, hyper parameter tuning.

Numerical Analysis

Optimization algorithms

Demo : training a logistic regression model.

1. Target distribution

$$p(y_i|\boldsymbol{x}_i) = \mathcal{B}(y_i|\pi_i)$$

2. Model construction

$$\pi_i = \sigma(\boldsymbol{x}_i^T \boldsymbol{w})$$

3. Objective function :

$$\mathcal{L}_i = f(y_i|\pi_i) = \pi_i^{y_i}(1-\pi_i)^{\mathbf{1}-y_i}$$

$$\mathcal{L} = \prod_{i=\mathbf{1}}^{N} \pi_i^{y_i}(1-\pi_i)^{\mathbf{1}-y_i}$$

$$\ell = \sum_{i=\mathbf{1}}^{N} y_i \log(\pi_i) + (1-y_i)\log(1-\pi_i) = -J(\boldsymbol{w})$$

4. First and second order derivatives :

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_{i=\mathbf{1}}^{N}(y_i - \pi_i)\boldsymbol{x}_i = \boldsymbol{X}^T(\boldsymbol{y}-\boldsymbol{\pi}) \in \mathbb{R}^p$$

$$\frac{\partial^{\mathbf{2}} J(\boldsymbol{w})}{\partial \boldsymbol{w}\partial \boldsymbol{w}^T} = \sum_{i=\mathbf{1}}^{N} -\pi_i(1-\pi_i)\boldsymbol{x}_i\boldsymbol{x}_i^T \in \mathbb{R}^{p\times p} = -\boldsymbol{X}^T diag(\boldsymbol{\pi}(1-\boldsymbol{\pi}))\boldsymbol{X} \in \mathbb{R}^{p\times p}$$

Numerical Analysis

Optimization algorithms

Newton Raphson :

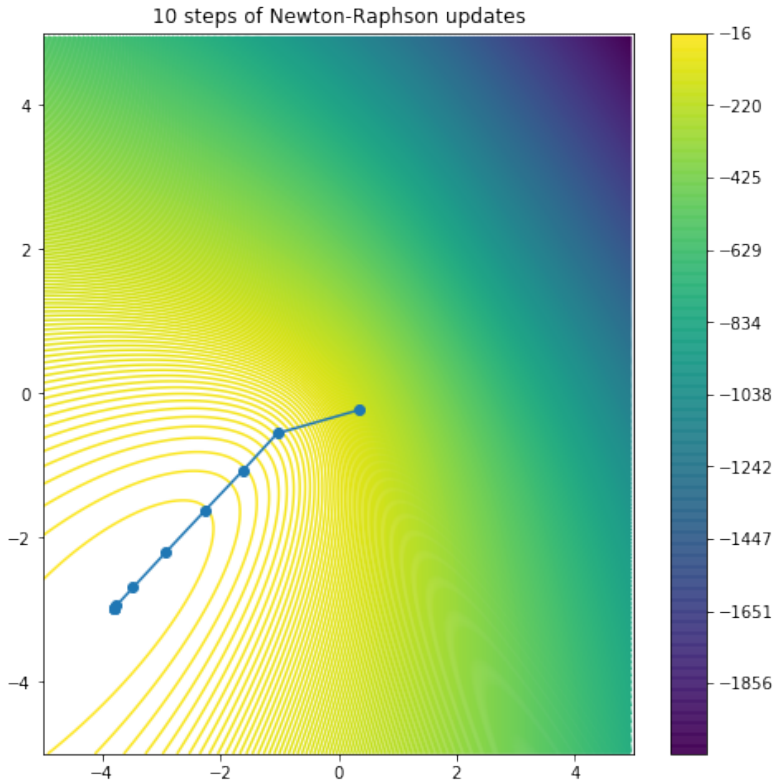


Figure – 10 steps of Newton-Raphson updates.

Numerical Analysis

Optimization algorithms

Gradient descent :

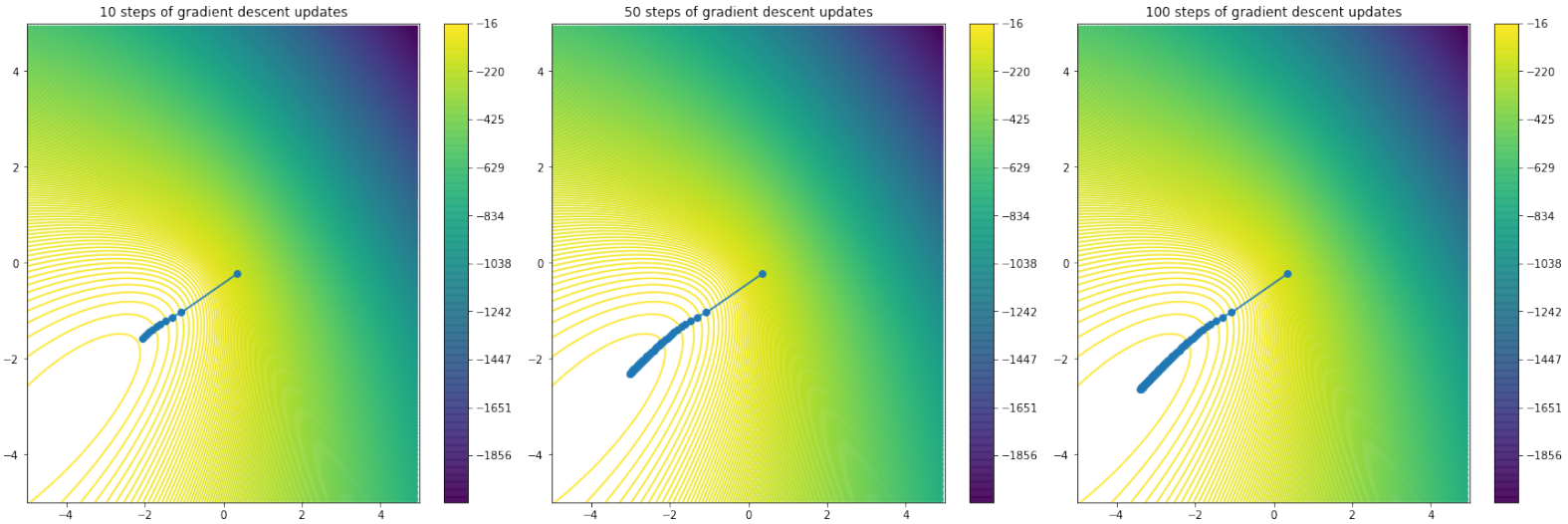
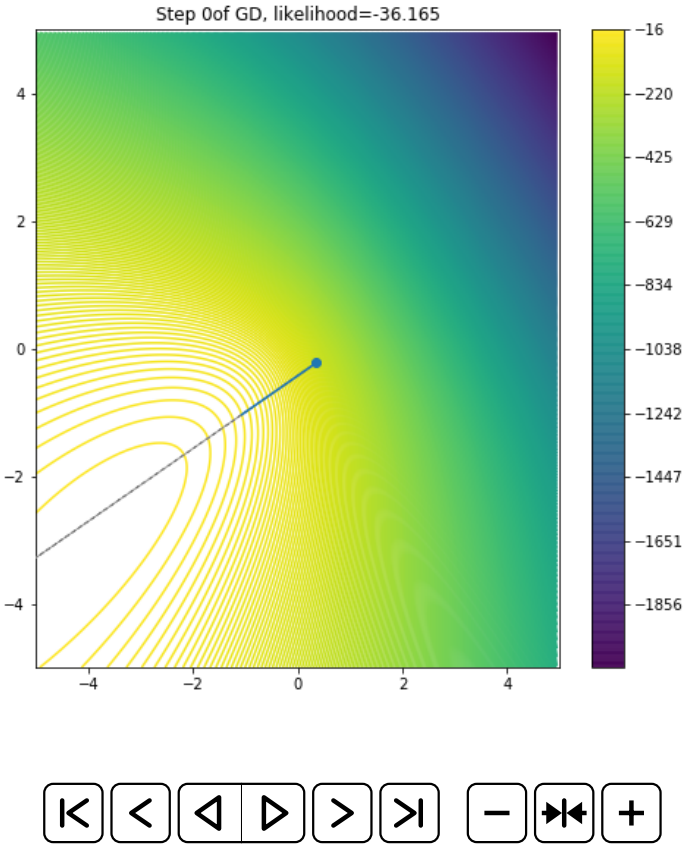


Figure – 10, 50 and 100 steps of gradient descent updates.

Numerical Analysis

Optimization algorithms

All gradient descent steps :



Numerical Analysis

Optimization algorithms

Evolution strategy :

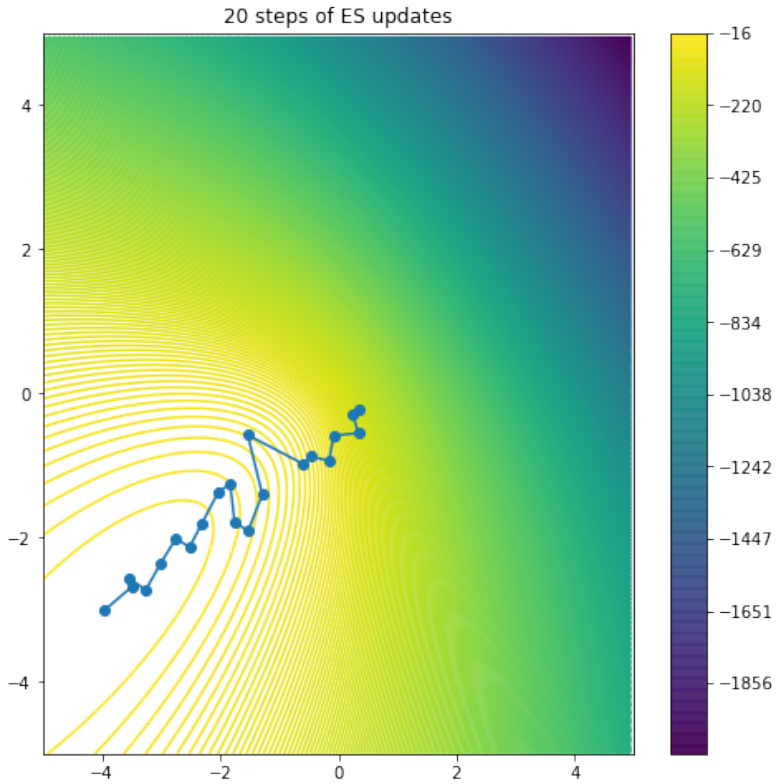
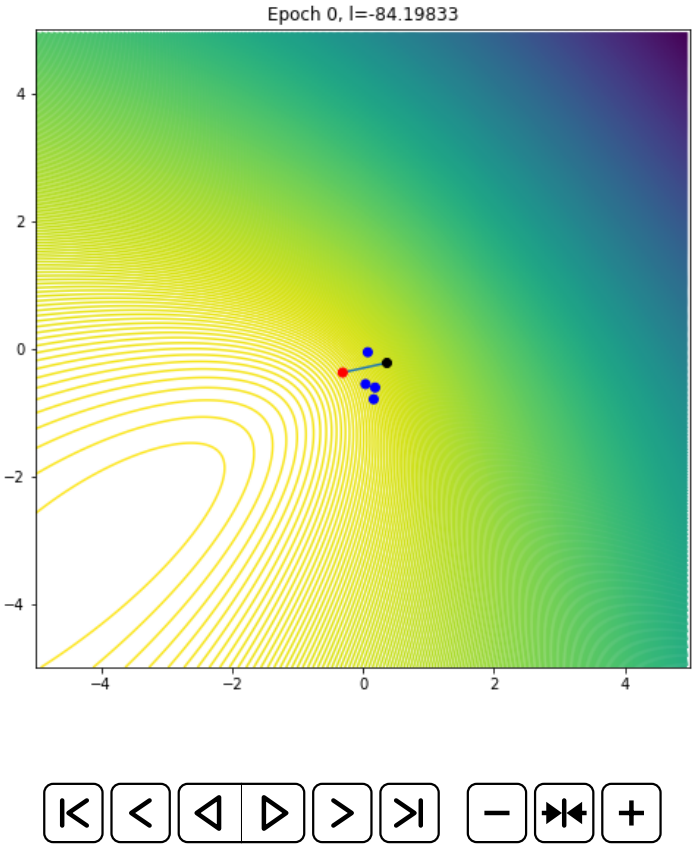


Figure – 20 steps of evolution strategy updates.

Numerical Analysis

Optimization algorithms

All ES steps :



Numerical Analysis

Automatic Differentiation

Automatic Differentiation could be ambiguous :

- Numerical differentiation : $f'(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$.
  - Known issue : rounding/truncation error, multiple evaluations.
- Symbolic differentiation : register a derivative for each math expression to simplify the end expression (mathematica).
  - Known issue : expression swell.
- Automatic differentiation : register a derivative for each primitive operation. (Theano, Tensorflow, Torch...)

Numerical Analysis

Automatic Differentiation

Foundations of AD (in strict sense) :

- Any function is evaluated by performing a sequence of elementary operations.
- The chain rule of differentiation. Given, e.g., $v_3 = f_3(v_2), v_2 = f_2(v_1), v_1 = f_1(x)$, we have

$$\frac{\partial v_3}{\partial x} = \frac{\partial v_3}{\partial v_2}\frac{\partial v_2}{\partial v_1}\frac{\partial v_1}{\partial x}$$

More generally, if $\boldsymbol{x} \in \mathbb{R}^p$, $\boldsymbol{v}_1 \in \mathbb{R}^{m_1}$, $\boldsymbol{v}_2 \in \mathbb{R}^{m_2}$, $\boldsymbol{v}_3 \in \mathbb{R}^{m_3}$, then

$$\frac{\partial \boldsymbol{v}_3}{\partial x} = \underbrace{\frac{\partial \boldsymbol{v}_3}{\partial \boldsymbol{v}_2}}_{m_3 \times m_2}\ \underbrace{\frac{\partial \boldsymbol{v}_2}{\partial \boldsymbol{v}_1}}_{m_2 \times m_1}\ \underbrace{\frac{\partial \boldsymbol{v}_1}{\partial \boldsymbol{x}}}_{m_1 \times p}$$

Numerical Analysis

Automatic Differentiation

Two basic modes of AD, forward and reverse modes, are different in how to accumulate

$$\frac{\partial v_3}{\partial x} = \frac{\partial v_3}{\partial v_2}\frac{\partial v_2}{\partial v_1}\frac{\partial v_1}{\partial x}$$

in an algorithmic way.

Step	Forward Mode	Reverse Mode
1	$\frac{\partial v_1}{\partial x}$	$\frac{\partial v_3}{\partial v_2}$
2	$\frac{\partial v_2}{\partial v_1}\frac{\partial v_1}{\partial x}$	$\frac{\partial v_3}{\partial v_2}\frac{\partial v_2}{\partial v_1}$
3	$\frac{\partial v_3}{\partial v_2}\frac{\partial v_2}{\partial v_1}\frac{\partial v_1}{\partial x}$	$\frac{\partial v_3}{\partial v_2}\frac{\partial v_2}{\partial v_1}\frac{\partial v_1}{\partial x}$

Numerical Analysis

Automatic Differentiation

Given vector input $\boldsymbol{x} \in \mathbb{R}^p$, vector output $\boldsymbol{y} \in \mathbb{R}^q$ and k intermediate primitive operation steps, i.e. $\boldsymbol{v}_i = f_i(\boldsymbol{v}_{i-1}), i \in [1, k]$, the forward mode calculates in one sweep [4] :

$$\text{for } i = 1 \ldots k :$$

$$\boldsymbol{v_i} = f_i(\boldsymbol{v}_{i-1}) \in \mathbb{R}^{m_i}$$

$$\dot{\boldsymbol{v}}_i = \frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{x}^T} = \underbrace{\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{v}_{i-1}}}_{m_i \times m_{i-1}} \underbrace{\dot{\boldsymbol{v}}_{i-1}}_{m_{i-1} \times p} \in \mathbb{R}^{m_i \times p}$$

At each step i :

- v_i and $\dot{v}_i$ ("tangent") are calculated simultaneously, and can be also implemented with dual numbers.
- Complexity $\mathcal{O}(m_*^2 \cdot p)$.
- Needs to store a matrix of shape $m_i \times p$ at step i.

4. Nocedal and Wright, 2006, section 8.2, p204-207

Numerical Analysis

Automatic Differentiation

A dual number in general :

$$w = a + b \cdot \epsilon$$

with $a, b \in \mathbb{R}$, $\epsilon > 0, \epsilon^2 = 0$, consists a real part a and a dual part $b \cdot \epsilon$.
Further,

$$\begin{aligned}(a + b \cdot \epsilon) + (c + d \cdot \epsilon) &= (a + c) + (b + d) \cdot \epsilon \\ (a + b \cdot \epsilon)(c + d \cdot \epsilon) &= ac + (ad + bc) \cdot \epsilon \\ \frac{a + b \cdot \epsilon}{c + d \cdot \epsilon} &= \frac{a + b \cdot \epsilon}{c + d \cdot \epsilon} \cdot \frac{c - d \cdot \epsilon}{c - d \cdot \epsilon} \\ &= \frac{a}{c} + \frac{bc - ad}{c^2} \cdot \epsilon\end{aligned}$$

Taking the exact Taylor expansion :

$$\begin{aligned}f(w) = f(a + b \cdot \epsilon) &= f(a) + \frac{f'(a)}{1!}(b \cdot \epsilon) + \frac{f''(a)}{2!}(b \cdot \epsilon)^2 + \frac{f'''(a)}{3!}(b \cdot \epsilon)^3 + \ldots \\ &= f(a) + \frac{f'(a)}{1!}(b \cdot \epsilon) + 0\end{aligned}$$

For $b = 1$ (what we are interested in) :

$$f(a + \epsilon) = f(a) + f'(a) \cdot \epsilon$$

Numerical Analysis

Automatic Differentiation

Examples :

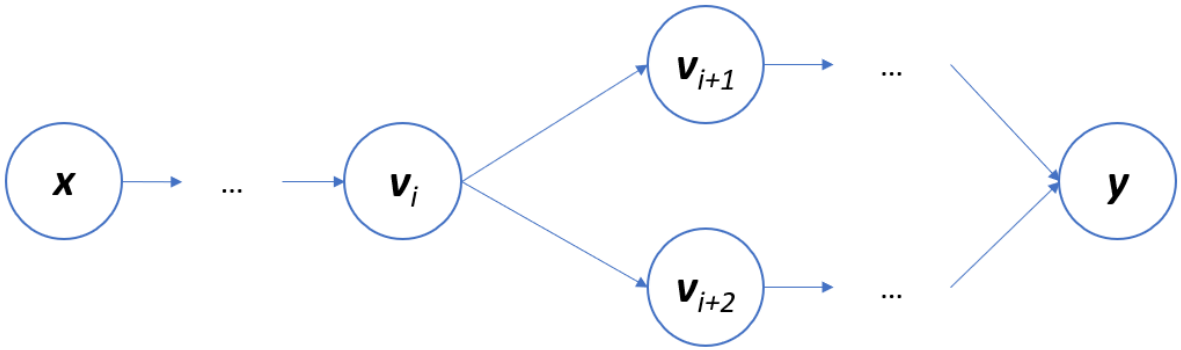
$$\begin{aligned} f(a) &= 3a^2 + 2a + 1 \\ &:= 3 \cdot (a + \epsilon)^2 + 2 \cdot (a + \epsilon) + 1 \\ &= 3a^2 + 2a + 1 + (6a + 2) \cdot \epsilon \\ &= f(a) + f'(a) \cdot \epsilon \end{aligned}$$

$$\begin{aligned} f(\boldsymbol{w}) &= \boldsymbol{w}^T \boldsymbol{x} \\ &:= (\boldsymbol{w} + \epsilon)^T \boldsymbol{x} \\ &= \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{x} \cdot \epsilon \\ &= f(\boldsymbol{w}) + f'(\boldsymbol{w}) \cdot \epsilon \end{aligned}$$

Numerical Analysis

Automatic Differentiation

Reverse mode automatic differentiation requires a computation graph.



- Node : variables including input, output or intermediate.
- Edge / arrow : a function.
- Parent / Child : an arrow from parent node(s) to child node(s).

Numerical Analysis

Automatic Differentiation

Given vector input $\boldsymbol{x} \in \mathbb{R}^p$, vector output $\boldsymbol{y} \in \mathbb{R}^q$ and k intermediate primitive operation steps, i.e. $\boldsymbol{v}_i = f_i(\boldsymbol{v}_{i-1}), i \in [1, k]$, the reverse mode calculates in two sweeps [5] :

$$\text{for } i = 1 \ldots k:$$

$$\boldsymbol{v}_i = f_i(\boldsymbol{v}_{i-1}) \in \mathbb{R}^{m_i}$$

$$\text{for } i = k \ldots 1:$$

$$\bar{\boldsymbol{v}}_i = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{v}_i^T} = \sum_{j \in \text{child}(i)} \underbrace{\bar{\boldsymbol{v}}_j}_{q \times m_j} \underbrace{\frac{\partial \boldsymbol{v}_j}{\partial \boldsymbol{v}_i^T}}_{m_j \times m_i} \in \mathbb{R}^{q \times m_i}$$

- In contrast to forward mode, the derivatives ("adjoint") are calculated and accumulated in the second sweep.
- Complexity $\mathcal{O}(m_*^2 \cdot q)$.
- Needs to store a matrix of shape $q \times m_i$ at step i.
- The child operation requires the storage of the computation graph.

5. Nocedal and Wright, 2006, section 8.2, p207-210

Numerical Analysis

An example of reverse mode

A logistic regression (again)

$$\pi = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}$$

$$J = y \log(\pi) + (1 - y) \log(1 - \pi)$$

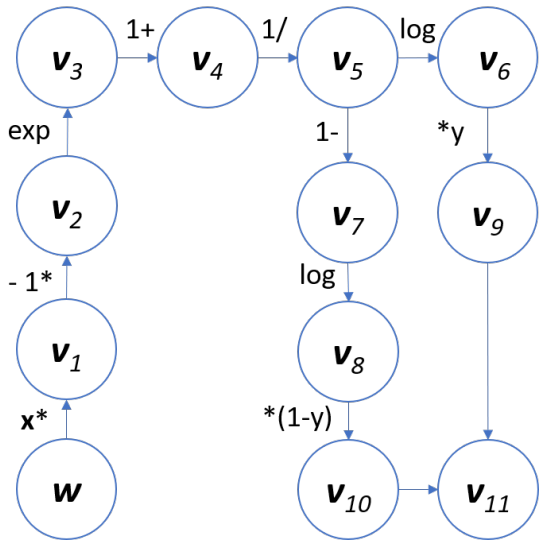


Figure – The computation graph

First sweep given :

- Current parameters $\boldsymbol{w} = (0.1, 0.2)^T$
- Training sample $(\boldsymbol{x} = (3.1, 4.5)^T, y = 1)$

$$\begin{aligned}
v_1 &= \boldsymbol{w}^T \boldsymbol{x} && = 1.21 \\
v_2 &= -v_1 && = -1.21 \\
v_3 &= \exp(v_2) && = 0.2982 \\
v_4 &= 1 + v_3 && = 1.2982 \\
v_5 &= 1/v_4 && = 0.7703 \\
v_6 &= \log(v_5) && = -0.2610 \\
v_7 &= 1 - v_5 && = 0.2297 \\
v_8 &= \log(v_7) && = -1.471 \\
v_9 &= y \cdot v_6 && = -0.2610 \\
v_{10} &= (1 - y) \cdot v_8 && = 0 \\
v_{11} &= v_9 + v_{10} && = -0.2610 \\
J &= v_{11}
\end{aligned}$$

Numerical Analysis

An example of reverse mode cont'd

Second sweep :

$$
\begin{aligned}
\bar{v}_{11} &= \frac{\partial J}{\partial v_{11}} = 1 \\
\bar{v}_{10} &= \bar{v}_{11} \cdot \frac{\partial v_{11}}{\partial v_{10}} && = \bar{v}_{11} \cdot 1 && = 1 \\
\bar{v}_{9} &= \bar{v}_{11} \cdot \frac{\partial v_{11}}{\partial v_{9}} && = \bar{v}_{11} \cdot 1 && = 1 \\
\bar{v}_{8} &= \bar{v}_{10} \cdot \frac{\partial v_{10}}{\partial v_{8}} && = \bar{v}_{10} \cdot (1 - y) && = 0 \\
\bar{v}_{7} &= \bar{v}_{8} \cdot \frac{\partial v_{8}}{\partial v_{7}} && = \bar{v}_{8} \cdot \frac{1}{v_7} && = 0 \\
\bar{v}_{6} &= \bar{v}_{9} \cdot \frac{\partial v_{9}}{\partial v_{6}} && = \bar{v}_{9} \cdot y && = 1 \\
\bar{v}_{5} &= \bar{v}_{6} \cdot \frac{\partial v_{6}}{\partial v_{5}} + \bar{v}_{7} \cdot \frac{\partial v_{7}}{\partial v_{5}} && = \bar{v}_{6} \cdot \frac{1}{v_5} + \bar{v}_{7} \cdot (-1) && = 1.2982 \\
\bar{v}_{4} &= \bar{v}_{5} \cdot \frac{\partial v_{5}}{\partial v_{4}} && = \bar{v}_{5} \cdot (-v_4^{-2}) && = -0.7703 \\
\bar{v}_{3} &= \bar{v}_{4} \cdot \frac{\partial v_{4}}{\partial v_{3}} && = \bar{v}_{4} \cdot 1 && = -0.7703 \\
\bar{v}_{2} &= \bar{v}_{3} \cdot \frac{\partial v_{3}}{\partial v_{2}} && = \bar{v}_{3} \cdot \exp(v_2) && = -0.2297 \\
\bar{v}_{1} &= \bar{v}_{2} \cdot \frac{\partial v_{2}}{\partial v_{1}} && = \bar{v}_{2} \cdot (-1) && = 0.2297 \\
\bar{v}_{0} &= \bar{v}_{1} \cdot \frac{\partial v_{1}}{\partial w} && = \bar{v}_{1} \cdot x && = (0.7121, 1.0337)^T
\end{aligned}
$$

Numerical Analysis

An example of reverse mode cont'd

If we expand the last expression symbolically :

$$
\begin{aligned}
\bar{v}_0 &= \bar{v}_1 \cdot \boldsymbol{x} \\
&= \left(\frac{y}{v_5} - \frac{1-y}{v_7} \right) \cdot (-v_4^{-2}) \cdot \exp(v_2) \cdot (-1) \\
&= \left(\frac{y}{\pi} - \frac{1-y}{1-\pi} \right) \cdot (1 + \exp(-\boldsymbol{w}^T \boldsymbol{x}))^{-2} \cdot (\exp(-\boldsymbol{w}^T \boldsymbol{x})) \cdot \boldsymbol{x} \\
&= \frac{y-\pi}{\pi(1-\pi)} \cdot (1-\pi) \cdot \pi \cdot \boldsymbol{x} \\
&= (y-\pi) \cdot \boldsymbol{x} \\
&= (0.7121, 1.0337)^T
\end{aligned}
$$

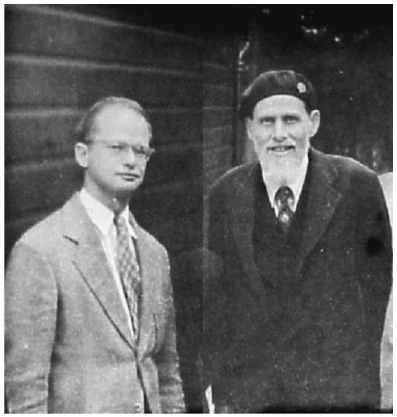
Try writing down the forward mode and show that the accumulated derivative is the same :)

Math Primer
Summary

- Selected topics from probability theory :
  - Maximum Likelihood Estimation
  - Maximum A Posteriori Esitmation : MLE and Prior
  - For supervised learning : probabilistic frameworks to formulate the learning objective.
- Numerical analysis
  - Optimization algorithms : Gradient descent as the dominating method in training neural networks due to *the large number of parameters*.
  - Automatic differentiation : Reverse mode of autodiff as the preferred method due to *the large number of parameters*.

Table of Contents

McCulloch and Pitts, 1940s



McCulloch (right) and Pitts (left) in 1949

- Warren McCulloch and Walter Pitts' paper :

  A logical calculus of the ideas immanent in nervous activity

- Model :

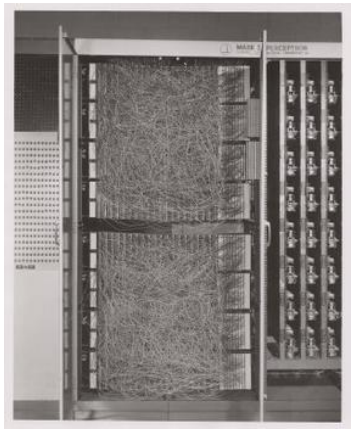
$$\hat{y} = \mathbb{1}(\sum_{j=1}^{p} x_j \geq \theta)$$

  with $x_j \in \{0, 1\}\ \forall j, \theta \in \mathbb{R}$

- Can model a variety of logic operations including AND, OR, NOR, by hard coding θ.
- Limitations :
  - All inputs are binary.
  - All inputs have equal contribution.
  - Cannot represent XOR.
  - No learning rules.

Rosenblatt, 1950s

Frank Rosenblatt



The Mark I Perceptron

- Known as the Single-Layer Perceptron.

  The Perceptron –A Perceiving and Recognizing Automaton

- Model :

$$\hat{y} = \mathbb{1}(\boldsymbol{\beta}^T \boldsymbol{x} + \beta_0 \geq 0)$$

  with $\boldsymbol{x} \in \{0,1\}^p, \boldsymbol{\beta} \in \mathbb{R}^p, \beta_0 \in \mathbb{R}$

- Not only a mathematical model, but an actual machine.

Rosenblatt, 1950s

- The learning algorithm :

$$\boldsymbol{\beta} := \boldsymbol{\beta} + \alpha \cdot (y_i - \hat{y}) \cdot \boldsymbol{x}$$
$$\beta_0 := \beta_0 + \alpha \cdot (y_i - \hat{y})$$

- Note the similarity to logistic regression :

$$\frac{\partial}{\partial \boldsymbol{\beta}} y \log(\pi) + (1 - y) \log(1 - \pi) = (y - \pi)\boldsymbol{x},$$
$$\text{where } \pi = \frac{1}{1 + \exp\left(-\boldsymbol{\beta}^T \boldsymbol{x}\right)}$$

- Indicator function $\leftarrow$ sigmoid function.
- Limitations : Only linearly separable tasks.
- Criticized by Marvin Minsky and Seymour Papert

Perceptrons : an introduction to computational geometry

- Followed by the AI winter starting 1960.

Intuition : XOR problem

- The XOR problem is only one of many linearly non-separable problems.

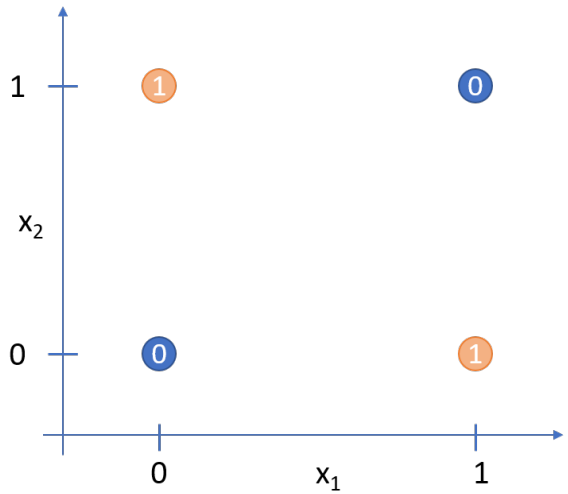


Figure – The XOR classification problem

- Dataset :

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Perceptrons
XOR Problem

- XOR is only solvable with two layers of basic logical operations ;
- logical operations can be realized as McCulloh-Pitts-Cell / Single-layered perceptrons $f(\boldsymbol{x}|\boldsymbol{\beta}, \beta_0)$:

$$AND(x_1, x_2) = f((x_1, x_2)|1, 1, 1.5)), OR(x_1, x_2) = f((x_1, x_2)|1, 1, 0.5))$$

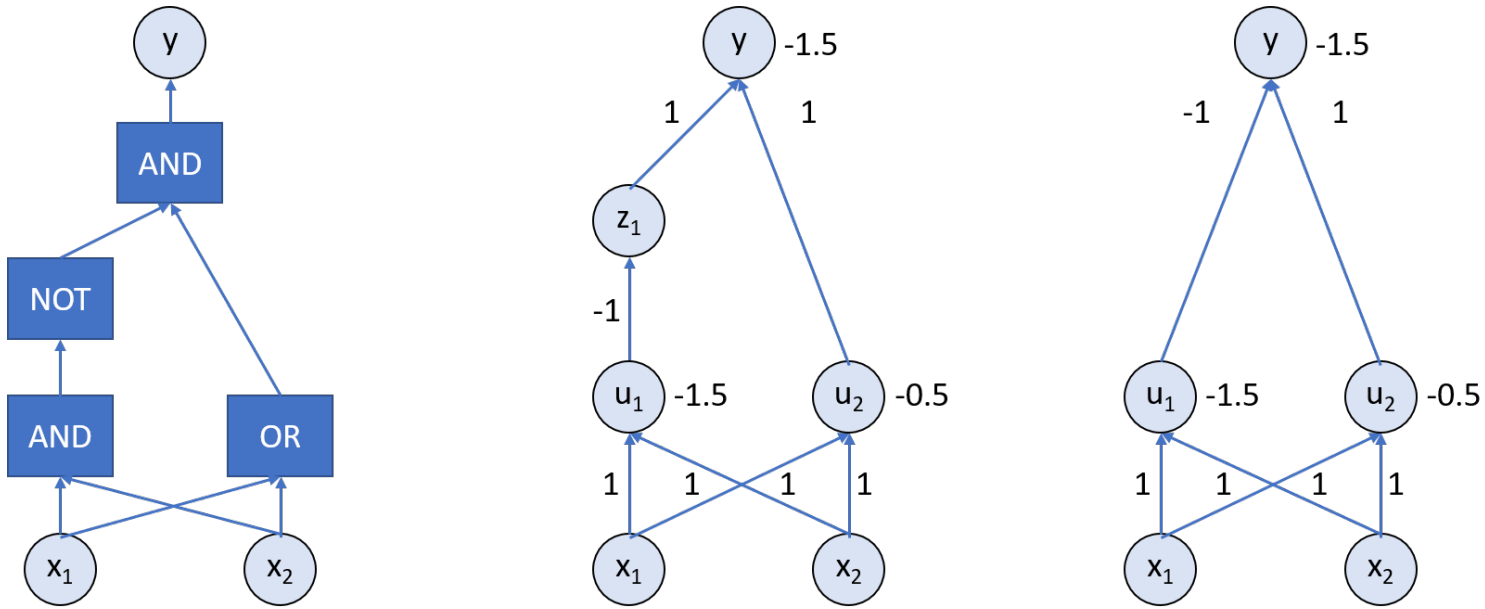


Figure – Left : Solving XOR with logical operations ; Middle : Perceptrons mimicing logical operations ; Right : Simplified solution with a new perceptron configuration : an MLP

Perceptrons
XOR Problem

What just happened ? ? [6]

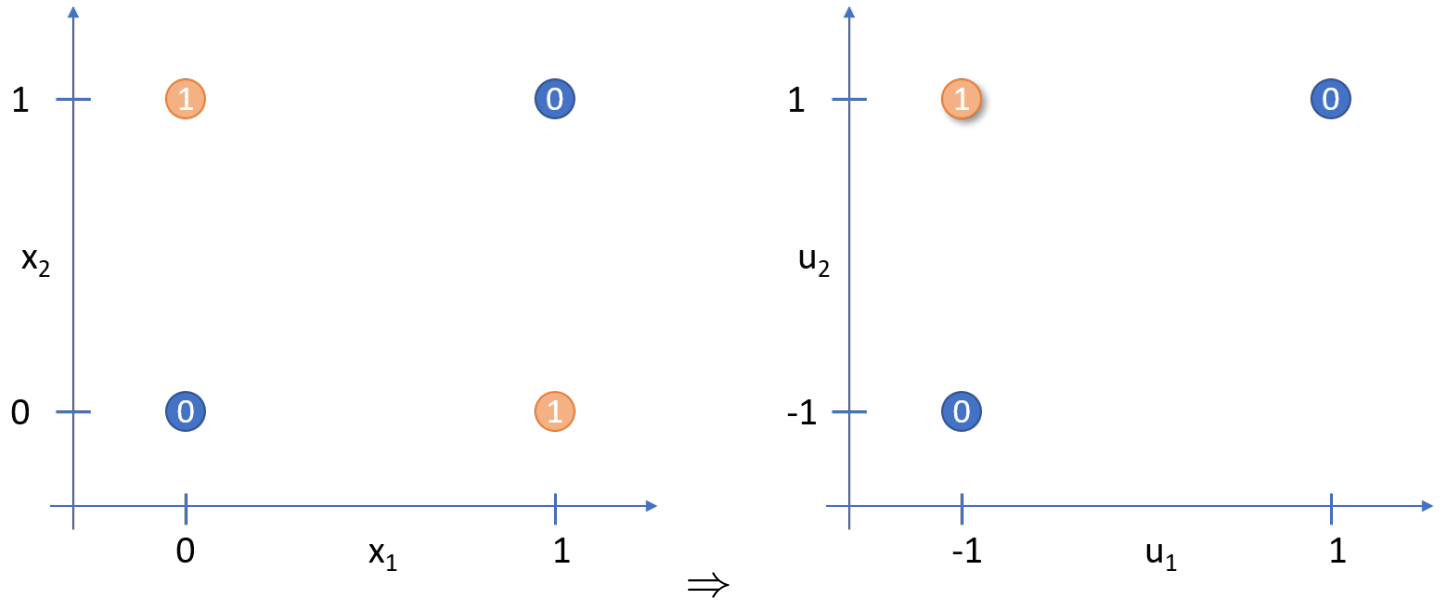


Figure – Transforming from x_1, x_2 to u_1, u_2

6. Goodfellow et al. 2016, section 6.1, p166

Perceptrons
XOR Problem

One has trained an MLP with $a(z) = \sigma(z)$ that solves the XOR problem. The weights are :

$$\boldsymbol{W} = \begin{pmatrix} -2.13 & -3.36 & 2.25 \\ -1.58 & 3.24 & -2.20 \end{pmatrix} \boldsymbol{u} = \begin{pmatrix} -0.65 \\ 2.11 \\ 2.47 \end{pmatrix}$$

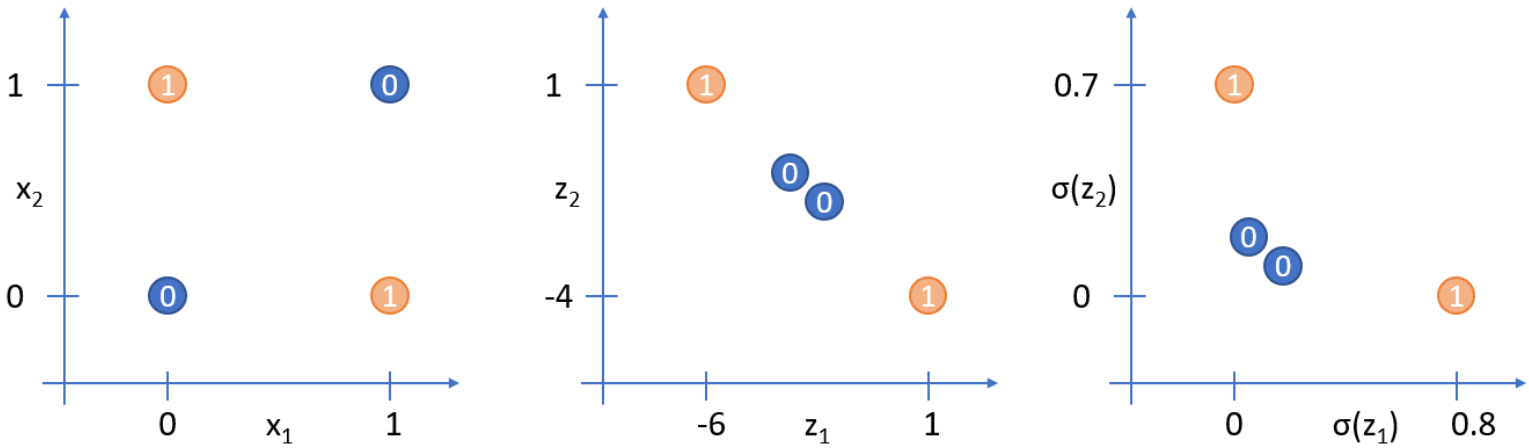


Figure – Left : values of x_1, x_2 ; Middle : values of z_1, z_2, still linear ; Right : values of $\sigma(z_1), \sigma(z_2)$

Perceptrons

Backpropagation, 1980s and 1990s

David Rumelhart

Vladimir Vapnik

- Werbos 1982, one of the first papers applying the chain rule to train NNs with multiple layers :

  Applications of advances in nonlinear sensitivity analysis

- Rumelhart et al. 1986, showing that the BP encourage hidden units to encode relevant information in the input :

  Learning representations by back-propagating errors

- LeCun et al. 1989, proposed to apply BP to train two layers of convolutioal network :

  Backpropagation Applied to Handwritten Zip Code Recognition

- MLP outperformed by the extended Support Vector Machines (Cortes and Vapnik, 1993/1995) :

  Support-Vector Networks

  that addresses the *soft margin* and the *kernel trick*.

Modern Neural Networks

RNN and the vanishing gradients, 1990s

Jürgen Schmidhuber

Sepp Hochreiter

- Hochreiter 1991, identified the cause of difficult in training neural networks of multiple layers being the "vanishing gradient problem" :

  Untersuchungen zu dynamischen neuronalen Netzen

- Hochreiter and Schmidhuber 1997, proposed an improvement to RNN :

  Long Short-Term Memory

Intuition : Vanishing Gradient

Based on Hochreiter et al. 2001 :

- Given a neural network of L hidden layers with element-wise sigmoid function as activation :

$$\boldsymbol{a}^{[l]} = \sigma(\boldsymbol{z}^{[l]}) = \frac{1}{1+\exp(-\boldsymbol{z}^{[l]})}, \ \boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l-1]}\boldsymbol{a}^{[l-1]} \ \forall l \in [1, L]$$

- The gradient w.r.t. $\boldsymbol{W}^{[l_0]}, l_0 \in [1, L]$ consists of multiplications of Jacobians :

$$\frac{\partial J}{\partial \boldsymbol{W}^{[l_0]}} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial \boldsymbol{a}^{[L]}}\frac{\partial \boldsymbol{a}^{[L]}}{\partial \boldsymbol{a}^{[L-1]}}\frac{\partial \boldsymbol{a}^{[L-1]}}{\partial \boldsymbol{a}^{[L-2]}}\cdots\frac{\partial \boldsymbol{a}^{[l_0+1]}}{\partial \boldsymbol{W}^{[l_0]}}$$

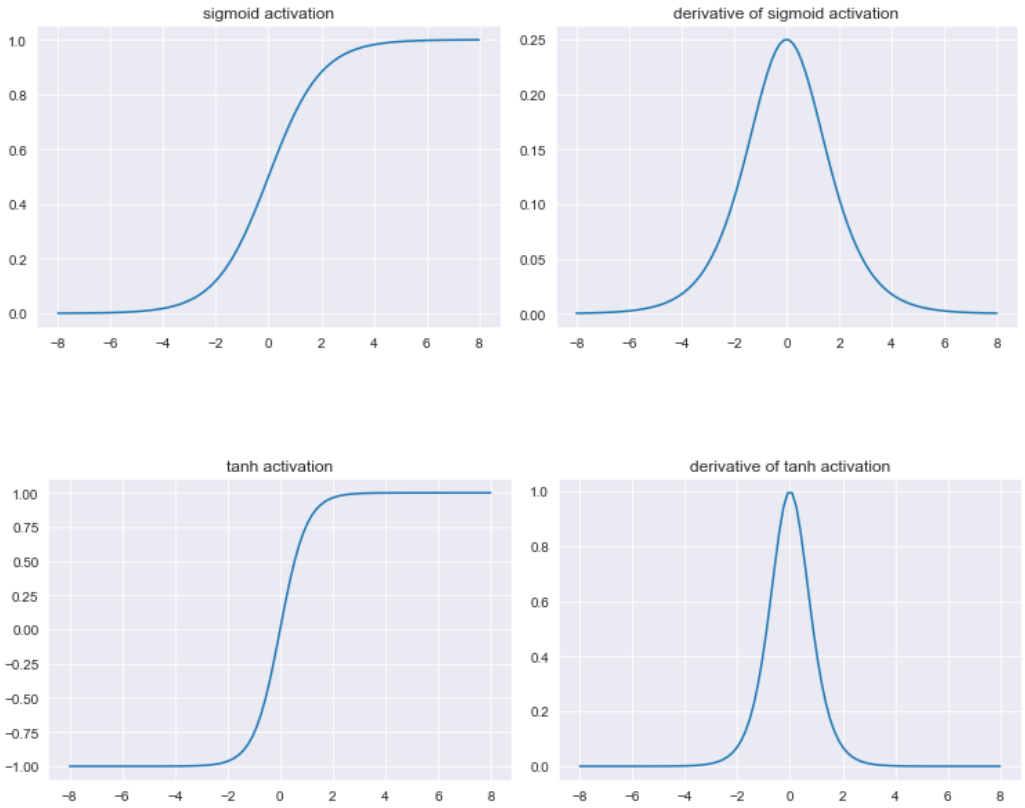
- For any $l > l_0$ we have

$$\frac{\partial \boldsymbol{a}^{[l]}}{\partial \boldsymbol{a}^{[l-1]}} = \frac{\partial \boldsymbol{a}^{[l]}}{\partial \boldsymbol{z}^{[l]}}\frac{\partial \boldsymbol{z}^{[l]}}{\partial \boldsymbol{a}^{[l-1]}} \text{ with } \frac{\partial \boldsymbol{z}^{[l]}}{\partial \boldsymbol{a}^{[l-1]}} = \boldsymbol{W}^{[l-1]} \text{ and}$$

$$\frac{\partial \boldsymbol{a}^{[l]}}{\partial \boldsymbol{z}^{[l]}} = \frac{\partial \sigma(\boldsymbol{z}^{[l]})}{\partial \boldsymbol{z}^{[l]}} = \sigma(\boldsymbol{z}^{[l]})(1-\sigma(\boldsymbol{z}^{[l]})) \leq 0.25$$

since $arg\max_{\sigma} \sigma(1-\sigma) = 0.5$

Intuition : Vanishing Gradient



Modern Neural Networks

CNN and the breakthrough of deep learning

Yann LeCun

Yoshua Bengio

- LeCun et al. 1998, showed that convolutional neural network outperforms other methods in character recognition task :

  Gradient-based learning applied to document recognition

- Bengio et al. 2007 on pre-training deep neural entworks.

  Greedy layer-wise training of deep networks

- Krizhevsky et al. 2012 halved the existent error rate with CNN, Relu and GPU.

  ImageNet Classification with Deep Convolutional Neural Networks

Modern Neural Networks

Summary

- McCulloh-Pitts Cell.
- Rosenblatt's Perceptron and XOR problem.
- Werbos and Rumelhart on Backpropagation.
- Hochreiter and Schmidhuber on vanishing gradient problem.
- Success of AlexNet 2012 and deep learning.

- XOR problem $\rightarrow$ nonlinear activation $\rightarrow$ vanishing gradient problem $\rightarrow$ initialization, activation, GPU, ... $\rightarrow$ explainability, reliability, robustness, ethics, energy ?

References

Textbooks

- *Pattern Recognition and Machine Learning*, Christopher M. Bishop, 2006.
- *Numerical Optimization*, Jorge Nocedal and Stephen J. Wright, 2006
- *Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016*

Papers

- Werbos, Paul J. "Applications of advances in nonlinear sensitivity analysis." System modeling and optimization. Springer, Berlin, Heidelberg, 1982. 762-770.
- Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." nature 323.6088 (1986) : 533-536.
- LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." Neural computation 1.4 (1989) : 541-551.
- Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995) : 273-297.
- Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997) : 1735-1780.
- LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998) : 2278-2324.
- Bengio, Yoshua, et al. "Greedy layer-wise training of deep networks." Advances in neural information processing systems. 2007.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012) : 1097-1105.
- Hochreiter, Sepp, et al. "Gradient flow in recurrent nets : the difficulty of learning long-term dependencies." (2001).