I_ Introduction

Supervised binary classification :

Gen approach :

How to model $p(x_i, y_i)$?

Naive Bayes, Generative / Hidden Markov Model

Discriminative approach

Which feature allow to distinguish ?
→ draw a boundary

Decision Trees , SVMs

II The formal neuron and Perceptron

Perceptron : $\hat{y}_i = \text{sign}(\omega^T x_i + b)$, $\theta = \{\omega, b\}$ parameters of the model

Learning procedure

Algo CV si les classes sont linéairement séparables
→ $\frac{R^2}{\gamma^2}$ it

Init : $\omega_t = 0$

Boucle : . $\forall i \in [1, n]$
- $\hat{y}_i = \text{sign}(\omega^T x_i)$
- Si $y_i \neq \hat{y}_i$: $\omega_{t+1} = \omega_t + \text{sign}(y_i) x$

Limites : → Quality of the boundary
→ Case of non-linear separability

Loss function :
- zero-one : $\mathbb{1}_{y - \text{sign}(f(x, \omega))}$
- hinge : $\max(0, 1 - y \cdot f(x, \omega))$
- MSE : $(y - f(x, \omega))^2$

Sigmoid : on remplace sign par $\sigma(x) = \frac{e^x}{1 + e^x}$ → différentiable

$\Rightarrow \hat{y}_i = \sigma(\omega^T x_i)$
↳ c'est maintenant une proba

Probabilistic loss function : MLE estimation

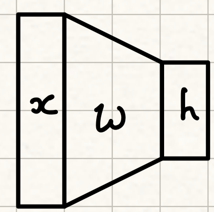
$$\ell_{MLE}(\theta) = -\log \mathcal{L}(\theta) \quad \text{avec} \quad \mathcal{L}(\theta) = \prod f(x_i, \omega)^{y_i} (1 - f(x_i, \omega))^{1-y_i}$$

Gradient : $\dfrac{\partial \ell_{MLE}}{\partial \omega^j} = -\sum [y_i - f(x_i, \omega)] x_i^j$

Gradient-based learning procedure :

- While $\| \ell_{MLE}(\omega_{t+1}) - \ell_{MLE}(\omega_t) \| > \delta$ et $t > m_{iter}$:
    - Pour $j \in [1, d]$:
      $$\omega_{t+1}^j = \omega_t^j + \varepsilon \sum_n^d [y_i - f(x_i, \omega)] x_i^j$$

III _ Multi-Layer Perceptron



Neurons are grouped into layers :

$$h^{(1)} = \phi^{(1)} (\omega^{(1)T} x + b^{(1)})$$

ϕ : fonction d'activation possiblement non-linéaire

↳ chaque layer est une fonction du layer précédent

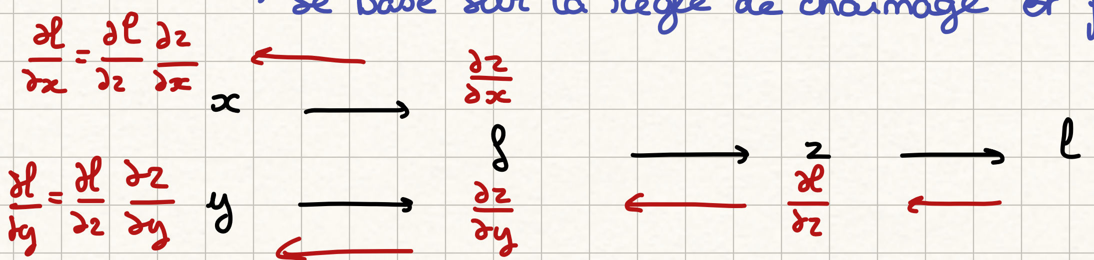
Activation : $\sigma(x) = \dfrac{1}{1+e^{-x}}$, $\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$

- **Softmax :** $\hat{o}_i = \left[\dfrac{e^{h^{(l),1}}}{\sum e^{h^{(l),k}}}, \dots \right]$

↳ généralisat° de la sigmoid pour k sorties

Optimisation : Backpropagation

→ se base sur la règle de chainage et fonctionne récursivement



IV - Model and Hyperparameter selection

→ Split data into train / test et choisis les hyperparamètres

↓ MIEUX

→ Split into train / validation / test
to tune h.p ↑ ↑ to verify perf

Choice of learning rate ε :

- Si perf ↗ sur validat° data : $\varepsilon = \varepsilon * \alpha$ $\quad \alpha \in [0,1]$

ou . Step decay : $\times \alpha$ tous les N epochs

ou . exponential decay : $\varepsilon^{(t)} = \varepsilon^{(0)} e^{-\alpha t}$

Regularisation : To avoid overfitting

$$\ell_{reg}(\omega) = \ell_{MLE}(\omega) + \lambda \omega^T \omega$$

$$\Rightarrow \omega^{t+1} = (1 - 2\lambda)\omega^t - \varepsilon \nabla_\omega \ell_{MLE}$$

Initialisation : dépend beaucoup des activat° & des méthodes d'optim

mais souvent $\omega^{(i)} \sim \mathcal{N}(0, \frac{1}{\sqrt{d_i}})$

Advantages :
→ very flexible in term of input / output
→ backpropagat° allows efficient updates
→ SGD allows efficient training $\bar{m}$ pr grosse data

Mais :
→ Gradient descent pas facile à executer
→ bcp d'hyperparamètres
→ risque overfitting (not enough data)
→ optimisat° hard → underfitting

Solution :
underfitting : more data
better optim

overfitting : find better way to regularise
→ unsupervised : init hidden layers with unsupervised learning

→ dropout : randomly remove parts of hidden layers

Better optimization : Adaptative gradient

Better activation : ReLU : $relu(x) = \max(0, x)$

→ avoid saturated & vanishing grad.

Better init : Unsupervised pre-training

↳ fine tuning = ajout d'un output et entraînement supervisé après cette init

Better reg : Dropout