

ADABOOST

- initialize data weights
 $\forall n, w_n^1 = \frac{1}{N}$
- for $t = [1, T]$:
 - a) find a classifier $y_t: x \rightarrow \{-1, 1\}$ that minimizes the weighted error
 $\sum_{n=1}^N y_t(x_n) w_n^t$
 - b) Evaluate

$$\epsilon_t = \frac{\sum_{n=1}^N y_t(x_n) w_n^t}{\sum_{n=1}^N w_n^t}$$

$$\alpha_t = \log\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$
 - c) Update the weights

$$w_n^{t+1} = w_n^t \exp(\alpha_t I(tn \neq y_t(\vec{x}_n)))$$

① normalisation
 $Y(\vec{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t y_t(\vec{x})\right)$

very sensitive to outliers (↑)
 training error goes down, test error goes up

Conjugate Gradient Descent

- start at \vec{x}_0
- $\vec{g}_0 = -\nabla f(\vec{x}_0)$
- for $k = 0, 1, \dots, n-1$:
 - find α_k that min $f(\vec{x}_k + \alpha_k \vec{g}_k)$
 - $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{g}_k$
 - $\beta_k = \frac{\|\nabla f(\vec{x}_{k+1})\|}{\|\nabla f(\vec{x}_k)\|}$
 - $\vec{g}_{k+1} = -\nabla f(\vec{x}_{k+1}) + \beta_k \vec{g}_k$
- set $\vec{x}_0 = \vec{x}_n$ and go to step 2 until convergence

① on utilise la moyenne / σ du training pour normaliser le validation set et test set.

DECISION TREE

- Neurone à quel point une classe est pure.
- Gini impurity: $\sum_{k=1}^C p_k(1-p_k)$ (nb de classes, proportion de la classe dans le nœud)
- + est proche de zéro + pur
- Shannon impurity: $-\sum_{k=1}^C p_k \log_2(p_k)$
- Gain = impurity du parent - (N gauche / N parent * impurity gauche + N droite / N parent * impurity droite)

RANDOM FORESTS

- ensemble ("bag") de trees construits sur des sous-échantillons aléatoires des données (bootstrap dataset) avec remplacement puis les faire voter (aggregate)
- à chaque nœud, on sélectionne les features possibles, pour éviter que les arbres se ressemblent
- out-of-bag datasets → on détecte de tous les arbres (on peut les utiliser comme validation set)

GRADIENT BOOSTED TREES

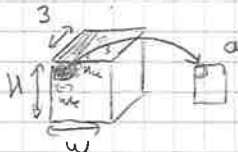
- on construit séquentiellement des trees pour corriger les erreurs des précédents
- ① $f_0(\vec{x}) = \arg \min_r \sum_{i=1}^N L(y_i, r)$ for $m = 1, \dots, M$
- $\vec{r}_m(\vec{x}_i)$ - compute residuals
- train tree on item $\rightarrow L(y_i, f_{m-1}(\vec{x}_i) + \vec{r}_m(\vec{x}_i))$
- $f_m(\vec{x}) = f_{m-1}(\vec{x}) + \eta \cdot h_m(\vec{x})$

① ne pas oublier de compter le bias dans le #nb de paramètres

NLP

- Retropagation: $\frac{\partial \mathcal{L}_0}{\partial w(x)} = \frac{\partial \mathcal{L}_0}{\partial z(x)} \cdot \frac{\partial z(x)}{\partial w(x)}$
- chaque layer prend le l'input
- forward pass: $\vec{z}^{(l)} = W^{(l)} \vec{a}^{(l-1)} + b^{(l)}$
- $\vec{a}^{(l)} = \sigma(\vec{z}^{(l)})$
- à la fin on a une softmax
- $E(\vec{w}) = \text{cross entropy loss multiclass}$
- $$= - \sum_{n=1}^N \sum_{k=1}^K y_{nk} \log \hat{y}_{nk}$$
- where $y_{nk} \in \{0, 1\}$ one-hot encoding du label

CNN

- $H_{out} = \left\lfloor \frac{H_{in} + 2P_{str} H_k}{S_H} \right\rfloor + 1$
- $W_{out} = \left\lfloor \frac{W_{in} + 2P_{str} W_k}{S_W} \right\rfloor + 1$
- bien penser que si image RGB et un kernel 2 en fait on a besoin de z_1, z_2, z_3 , donc
- 
- if we had x kernels at the layer we would have x feature maps at the output
- $Y(i,j,c) = \sigma \left[\sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \sum_{d=0}^{C_{in}-1} X[i+m, j+n, d] \cdot F[m, n, d, c] \right] + b_c$
- better generalization
- less padding
- translation invariant

pooling / striding → réduire les dims + champ de vision élargi

Transformers

① séparer l'input en "chunks" et transformer ces chunks (tokens) en embeddings (paires)

Embedding matrix: look-up-table entre le token/pair et embedding. taille $n \cdot \text{dim} \times n \cdot \text{tokens}$

① on ajoute un embedding de position à chaque embedding.

② Phase d'attention, partage de contexte entre tokens

Queries: les queries que le token se pose. On trace via W_q embedding

Keys: une série de FAQ préparée par l'embedding. — W_k embedding

Values: les réponses à chaque query de la FAQ.

On fait le produit entre query q_i et on fait un softmax pour que i capture des mots les plus intéressants.

embedding origin: $\text{softmax}(k^T Q) V$

masking: on cache les mots avant qu'ils ne soient utilisés pour répondre à la question. ça permet de faire du cross-attention: questions représentées de la même façon et les valeurs/keys de l'audio multi-head: plusieurs sets de questions, keys, values

③ NLP chaque neurone d'entrée pose des queries et en fonction de l'activité des neurones de sortie, on enrichit l'embedding

④ Unembedding matrix + softmax pour générer l'output