2- ml background

$\hat{y} = f(x;\theta)$ — output ← model ← learnable parameters, input

$L(\hat{y}, y; \lambda)$ — loss function ← output; hyper-parameters, ground truth

↳ even though they use non-linear activation functions

- linear models implements a linear decision boundary (ex: SVM with $w^T x$)
- non-parametric models almost everytime implements non-linear decision boundaries (ex: KNN)
- higher capacity → more complicated decision boundaries (high complexity can cause overfitting)

VC dimension = largest number the model can correctly label arbitrary configurations of n points, for binary classification. + / − ; + + / − − ; − − / + − + ✓ ; + − − + ✗

loss functions:

↳ 0-1 = counting how many mistakes $L(w) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}_{f(x_i;w) \neq y_i}$ → not used because not differentiable

↳ l2 = $L_2(w) = \sum_{i=1}^{N}(1-z_i)^2$ where $z_i = y_i \cdot f(x_i;w)$ • $\sum_{i=1}^{N}(y_i - f(x_i;w))^2$ in regression

↳ hinge = $L_{Hinge}(w) = \sum_{i=1}^{N} \max(0, 1-z_i)$ where $z_i = f(x_i;w)$ → SVM and max margin alg. uses

↳ log-loss = $L_{log}(w) = \sum_{i=1}^{N} \ln(1+e^{-z_i})$ where $z_i = y_i f(x_i;w)$ → similar to hinge but non-zero penalty to all

↳ cross-entropy = usually used with softmax

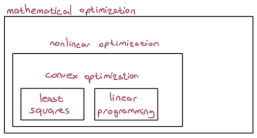
$f(s)_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$ } normalised softmax → $-\sum_i^N y_i \log(f(s)_i)$

- softmax is not a loss function, it is a normalization function

convexity = a func is convex if the line segment between any two distinct points on the function graph lies above the graph between two points

↳ if the second order derivative is positive, then convex

mathematical optimization



gradient descent = $w^{next} = w^{curr} - \eta \nabla_w L(w)$

derivative = rate of change

gradient = vector of partial derivatives

↳ $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$ → used in partial derivative

subgradient = substitude for gradient when it does not exist

gradient = the loss is calculated all the examples in the dataset, w updated accordingly

stochastic gradient = set of randomly chosen m example is used to update $w \in [1, N]$

↳ when the learning rate decreases, it converges almost surely

↳ when to stop → change in loss or parameters becomes to small

↳ performance in validation set does not improve

↳ max time or iteration is reached

regularization = controlling the capacity of a learning model → lower capacity is favored (↳ used to prevent overfitting)

↳ margin maximization in SVM → choosing best boundary to separate data

- $\underset{\text{cost}}{L(w)} = \underbrace{\frac{1}{2}\|w\|^2}_{\text{regularization}} + \underbrace{C\sum_i^N \max(0, 1-y_i w x_i)}_{\text{training error}}$ margin = $\frac{2}{\|w\|}$

↳ dropout and data augmentation is used in regularization

↳ L_2 and L_1 norm = $f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$ simpler → $\theta_0 + \theta_1 x_1$, if θ_2 and θ_3 small

cost = training error + $\lambda \sum_i |\theta_i|$ → L_1 regularization } simpler equations

loss func + $\lambda \sum_i (\theta_j)^2$ → L_2 regularization

model parameters = learned on training set → learning / training (with numerical optimizations like gradient descent)

hyper-parameters = learned on the validation set → model selection / tuning → with grid search, or etc

↳ model related = # layers in NN, C in SVM

↳ learning method related = learning rate, batch size..

* cross validation is used on both

maximum likelihood (ML) = $\underset{\theta}{\text{argmax}} \prod_{i=1} p(x^i|\theta) \longrightarrow \underset{\theta}{\text{argmax}} \sum_i^N \log(p(x_i;\theta))$ (to minimize, log both sides)

if this is uniform, ML & MAP are equivalent

maximum a-posteriori (MAP) = $\underset{\theta}{\text{argmax}} \prod_{i=1} p(x^i|\theta) \cdot \underbrace{p(\theta)}_{\text{prior}} \longrightarrow \underset{\theta}{\text{argmax}} \sum_i^N (\log(p(x_i;\theta)) + \log(p(\theta)))$

↳ both are used to estimate parameters