Synthesis

Supervised classification

- Statistics: Statistical inference, Consistency of the estimator, True risk estimation
- Optimization: Objective function, Constraints/penalties, Convergence/optimum
- Computer science: Algorithmic, Complexity in time and memory, Implementation issues
- Learning algorithm $\mathcal{A}$ that takes the training dataset and provide a function that classifies the data
    - Program that learns: `clf.fit(X_train, y_train)`
    - Program that predicts: `clf.predict(X_test, y_test)`
- Random feature vector X that takes its value in $\mathcal{X} = \mathbb{R}^p$
- Random output variable Y that takes its value in $\mathcal{Y}$
    - Regression: $\mathcal{Y} = \mathbb{R}$
    - Binary classification: $\mathcal{Y} = \{-1,1\}$
    - Multi-class classification: $\mathcal{Y} = \{1, \dots, C\}$
- A classifier is a type of algorithm or model in machine learning that is trained to assign labels or categories to input data based on its characteristics or features. In supervised learning, the goal of a classifier is to learn a mapping from input data to predefined output labels, enabling it to make predictions about new, unseen instances.
- A regressor is a type of algorithm or model in machine learning that is trained to predict continuous numerical values based on input features. Regression models aim to learn the relationship between input features and the target variable by fitting a function to the training data. This function can then be used to make predictions about new, unseen instances.
- Joint probability distribution: $P(X,Y)$, fixed but unknown
- Class of measurable functions from $\mathcal{X}$ to $\mathcal{Y} \subset \mathbb{R}$: $\mathcal{D}$
- Hypothesis space: $\mathcal{H} \subset \mathcal{D}$, the space of classification (regression) models
- (local) loss function: $\ell: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
- True risk (generalization error) of $h \in \mathcal{H}$ noted $R(h) = \mathbb{E}_{(X,Y)\sim P}[\ell(h(X),Y)]$
- Supervised learning: search for the solution of $\underset{h\in\mathcal{H}}{\operatorname{argmin}} \mathbb{E}_{(X,Y)}[\ell(h(X),Y)]$
    - With the help of the training sample $\mathcal{S}_n = \{(x_i, y_i)_{i=1}^n\}$ containing n identical independent realizations of (X,Y) and without knowledge of P
    - Provide an estimator $\hat{h}$ of the target function h^* leveraging a finite training sample
- Bayer rule: $P(Y=k|x) = \frac{P(x|Y=k)\cdot P(Y=k)}{P(x|Y=-1)P(Y=-1)+P(x|Y=1)P(Y=1)}$
- Bayes classifier: $h_{\text{Bayes}}(x) = \underset{c}{\operatorname{argmax}} P(Y=c|x)$
    - It can be shown that $h_{target} = h_{\text{Bayes}}$ and $R_{\text{Bayes}} = R(h_{\text{Bayes}})$ is the minimal risk associated to the zero-one loss
        - R_{Bayes} is characteristic of the "complexity" of the joint probability distribution P and the loss
- Statistical learning: define a learning algorithm $\mathcal{A}: S_n \to \mathcal{A}(S_n) \in H, \forall P, S_n \sim P, R(\mathcal{A}(S_n)) \to R(h_{target})$ in probability
- Empirical risk: $R_n(h) = \frac{1}{n}\sum_{i=1}^n \ell(h(x_i), y_i)$
    - Law of large numbers: when h is fixed, $R_n(h)$ tends towards $R(h)$ almost surely: $P(\lim_{n\to\infty}(R_n(h) = R(h))) = 1$
    - Statistical learning by empirical risk minimization: $\underset{h\in\mathcal{H}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^n \ell(h(x_i), y_i)$
        - The goal is to minimize the expected error on unseen data by minimizing the empirical risk, which is the average loss over the training dataset. This principle assumes that the training dataset is representative of the true underlying distribution of the data. By minimizing the empirical risk, machine learning algorithms aim to find the model parameters that best fit the training data and generalize well to unseen data. In practice, this often involves optimizing a loss function to find the parameters that minimize the discrepancy between the model predictions and the actual outcomes in the training data.
- Relevance of empirical risk minimization
    - Let $R_{\mathcal{H}} = \inf_{h\in\mathcal{H}} R(h)$ the smallest risk you can achieve in the function space $\mathcal{H}$.
    - Let $h_n \in \mathcal{H}$ be the classifier learnt from dataset S_n by minimization of the empirical risk or any method based on the dataset S_n.
    - $R(h_n) - R_{\text{Bayes}} = (R(h_n) - R_{\mathcal{H}}) + (R_{\mathcal{H}} - R_{\text{Bayes}})$
        - $R(h_n) - R_{\mathcal{H}}$: an estimation error that measures to which point h_n is close to the best solution in $\mathcal{H}$
        - $R_{\mathcal{H}} - R_{\text{Bayes}}$: an approximation error, inherent to the chosen class of functions
    - If $\mathcal{H}$ is too small, you cannot reach the target (large bias, no universality): risk of underfitting
        - Underfitting occurs when a machine learning model is too simplistic to capture the underlying patterns in the data. This often happens when the model is overly constrained, resulting in high bias. In such cases, the model fails to learn from the training data effectively, leading to poor performance not only on the training data but also on unseen data.
    - If $\mathcal{H}$ is too big, you cannot reduce variance (large variance, no consistency): risk of overfitting
        - Overfitting arises when a machine learning model captures noise or random fluctuations in the training data, rather than the underlying patterns. This typically occurs when the model is overly complex, leading to large variance. While an overfitted model may perform extremely well on the training data, it fails to generalize to new, unseen data because it has essentially memorized the noise in the training set.
- Supervised learning with regularization: $\underset{h\in\mathcal{H}}{\operatorname{argmin}} R_n(h) + \lambda\Omega(h), \Omega: \mathcal{H} \to \mathbb{R}^+$ to control complexity, more generally imposition of some prior knowledge
    - Pb1: $\underset{h\in\mathcal{H}}{\operatorname{argmin}} R_n(h), \Omega(h) \le C$
    - Pb2: $\underset{h\in\mathcal{H}}{\operatorname{argmin}} \Omega(h), R_n(h) \le C$
- Two families of approaches
    - Discriminative approaches: just find a classifier which discriminates with a separating hyperplane (decision trees, SVM)
    - Generative probabilistic approaches: build a plug-in estimator of $\hat{P}(Y=1|x)$ using $P(x|Y=1), P(x|Y=-1)$ and prior probabilities (Naive Bayes, Generative/Hidden Markov Models)
- Parametric modelling: Linear models, Neural models
- Non-parametric modeling: Local average models (k-neighbors, trees), Kernel models

Ensemble Methods

- Target functions for a given loss
    - Classification: 0/1 prediction loss: Bayes classifier $f_{\text{Bayes}}(x) = \underset{c}{\operatorname{argmax}} P(Y=c|x)$ is the best solution
    - Regression: squared difference loss $\ell(h(x), yy) = (y - h(x))^2$: regression function $f_{\text{reg}}(x) = \mathbb{E}(Y|x)$ is the best solution (minimal risk)
- Curse of dimensionality, as the number of dimensions grows, the amount of data we need to generalize accurately grows exponentially
- Ensemble methods: simpler, general way to handle Machine Learning problems
    - Heavily depends on the number of available data
    - Model itself contains the training data on which it is based (transparency, pre-training), which increases its interpretability
- Decision trees:
    - Belong to the family of models of the form: $f(x) = \sum_{\ell=1}^m \mathbb{I}(x \in \mathcal{R}_\ell) f_\ell$
    - In practice, the $\mathcal{R}_\ell$'s form a partition and are estimated/learned using the training dataset as well as the f_ℓ's.
    - f_ℓ only depends on regions $\mathcal{R}_\ell \subset \mathcal{X}$ in the input space and on training data in this region.
    - Classification into C classes is done by majority voting $f_\ell = \arg\max_{c=1,\dots,C} \hat{p}_{\ell c}$, with $\hat{p}_{\ell c} = \frac{1}{N_\ell}\sum_{x_i\in\mathcal{R}_\ell} \mathbb{I}(y_i = c)$, and N_ℓ is the number of training data falling into region $\mathcal{R}_\ell$
    - Regression is done with $f_\ell = \frac{1}{N_\ell}\sum_{x_i\in\mathcal{R}_\ell} y_i$
- Separator or split:
    - Continuous variable: $t_{j,s}(x) = \operatorname{sign}(x^j - s)$
    - Categorical variable with $\{v_1^j, \dots, v_K^j\}$: $t_{j,v_k}(x) = 1(x^j = v_k^j)$
- Recursive building algorithm for trees
    1. Let $\mathcal{S}$ the training dataset
    2. Build a root node
    3. At the current node, find the best binary separation defined by the split t to be applied to $\mathcal{S}$ such that $L(t, \mathcal{S})$ be minimal
    4. Associate the chosen separator $t(\hat{j}, \hat{s})$ to the current node and split the current training dataset into two datasets at right and left side, $\mathcal{S}_\ell$ and $\mathcal{S}_r$
    5. Build two child nodes: a node at left associated with $\mathcal{S}_\ell$, a node at right associated with $\mathcal{S}_r$
    6. Compute a stopping criterion on the left node: if it is satisfied, this node becomes a leaf, otherwise go to 3
    7. Compute a stopping criterion on the right node: if it is satisfied, this node becomes a leaf, otherwise go to 3
- Classification: $\min_{j,s}\left[\frac{N_l}{N} H(\mathcal{S} \cap \mathcal{R}_l(j,s)) + \frac{N_r}{N} H(\mathcal{S} \cap \mathcal{R}_r(j,s))\right]$, impurity criterion H
    - Misclassification: $H(S) = 1 - p_k(S)$
    - Cross-entropy: $H(S) = -\sum_{k=1}^C p_k(S) \log p_k(S)$
        - Kullback-Leibler divergence: $\text{KL}(p,q) = \sum p_i \log\frac{p_i}{q_i}$
            - $\min \text{KL} = \min \text{cross-entropy}$
    - Gini index: $H(S) = 1 - \sum_{k=1}^C p_k(S)^2$
    - Finding best split naively: $O(dn^2c)$
    - Incrementally finding splits: $O(dnc)$
        - Sort feature non decreasingly; compute H for each value incrementally; return candidate with minimum H
- Regression: $\min_{j,s}\left[\sum_{x_i,x_j\in\mathcal{S}\cap\mathcal{R}_l}(y_i - y_j)^2 + \sum_{x_i,x_j\in\mathcal{S}\cap\mathcal{R}_r}(y_i - y_j)^2\right]$
- Stopping criterions: maximal depth, minimal number of data is reached at a node, no improvement, …
    - Cross validation can be used to select that hyperparameters
    - Pruning
- Decomposition bias/variance: expected error $\mathbb{E}_S\mathbb{E}_{y|x}(y - f_S(x))^2 = \text{noise}(x) + \text{bias}^2(x) + \text{variance}(x)$
    - $\text{noise}(x) = \mathbb{E}_{y|x}\left((y - \mathbb{E}_{y|x}(y))^2\right)^2$: quantifies the error made by the Bayes model ($\mathbb{E}_{y|x}(y)$)
    - $\text{bias}^2(x) = (\mathbb{E}_{y|x}(y) - \mathbb{E}_S(f_S(x)))^2$: measures the difference between the minimal error (Bayes error) and the average model
    - $\text{variance}(x) = \mathbb{E}_S(f_S(x) - \mathbb{E}_S(f_S(x)))^2$: measures how much $f_S(x)$ varies from one training set to another
- Bagging = Bootstrap Aggregating: bias increases a bit but variance is divided by T
    - Assume we can generate several training independent samples $S_1, \dots, S_T$ from $P(x,y)$.
    - Algorithm:
        - For sample t in $\{1, \dots, T\}$, do
            - $\mathcal{B} \leftarrow$ bootstrap (random samples with replacement) sample from S_t
            - $f_t \leftarrow$ learn a model with $\mathcal{B}$
        - Return $f_{\text{bag}}(x) = \frac{1}{T}\sum_{t=1}^T f_t(x)$
    - Effective sample: each bootstrap sample will contain $1 - e^{-1} \approx 0.632$ of the original sample as $\mathcal{S}$
        - From the same distribution as S, not independent ∴ not identically distributed
    - Bootstrap model is more complex than a single model
- Random forest
    - Algorithm:
        - For sample t in $\{1, \dots, T\}$, do
            - $\mathcal{B} \leftarrow$ bootstrap (random samples with replacement) sample from S_t
            - $f_t \leftarrow$ learn a randomized tree (randomly keep $k = \sqrt{p}$ features) with $\mathcal{B}$
        - Return $f_{\text{bag}}(x) = \frac{1}{T}\sum_{t=1}^T f_t(x)$
    - Pros:
        - Fast, parallelizable and appropriate for a large number of features
        - Relatively easy to tune
    - Cons:
        - Overfitting if the size of the tree is too large
        - Interpretability is lost (however importance of feature can be measured)
- Out-of-bag samples: $\{\overline{\mathcal{S}}_n = \mathcal{S}_n - \mathcal{S}_n^t, t = 1, \dots, n_{\text{tree}}\}$, samples not selected by bootstrap
    - Out-of-bag error: evaluating at training time
    - Variable importance
        - Let $\{\overline{\mathcal{S}}_n^{t,j}, t = 1, \dots, n_{\text{tree}}\}$ be permuted out-of-bag-samples
        - $\hat{I}(X^j) = \frac{1}{n_{\text{tree}}}\sum_{t=1}^{n_{\text{tree}}} R_n(h_t, \overline{\mathcal{S}}_n^{t,j}) - R_n(h_t, \overline{\mathcal{S}}_n^{t})$
- Boosting
    - Weak learners: average training error is no more than 0.5
    - AdaBoost
        - Algorithm:
            - $H_0 = 0$
            - $w_i = \frac{1}{n}$
            - For t in $\{1, \dots, T\}$ do:
                - $h = \underset{h\in\mathcal{H}}{\operatorname{argmin}} \sum_{i:h(x_i)\neq y_i} w_i$
                - $\epsilon \leftarrow \sum_{i:h(x_i)\neq y_i} w_i$

            - $\alpha = \frac{1}{2}\log\frac{1-\epsilon}{\epsilon}$
            - $H_t = H_{t-1} + \alpha h$
            - $w_i \leftarrow \frac{w_i \exp(-\alpha h(x_i) y_i)}{2\sqrt{\epsilon(1-\epsilon)}}$
        - Return H_{t-1}
    - Stage-wise additive model: $H(x) = \sum_t \alpha_t\, h_t(x)$
        - Greedy approach
    - Forward stage-wise additive model for the exponential loss $L(y, h(x)) = \exp(-yh(x))$
- Gradient boosting
    - Algorithm
        - Initialize model with a constant value $H_0(x) = \underset{\gamma}{\operatorname{argmin}} = \sum_{i=1}^{n} L\,(y_i, \gamma)$
        - For t in $\{1, \dots, T\}$, do
            - $r_i = -\frac{\partial L(y_i, H_{t-1}(x_i))}{\partial H_{t-1}(x_i)}$
            - $h_t(x) = \operatorname{argmin} \sum_i^n r_i\, h(x_i)$
            - $\gamma_t = \underset{\gamma}{\operatorname{argmin}} \sum_i^n L\,(y_i, H_{t-1}(x_i) + \gamma h_t(x_i))$
            - $H_t(x) = H_{t-1}(x) + \gamma_t h_t(x)$
        - Return H_T
- Can overfit, early stopping could be an answer

Support Vector Machine

- Maximize geometrical margin: $\rho(w) = \frac{1}{\|w\|}$
- Primal: $\underset{w,b,\xi}{\min} \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} \xi_i = \underset{w,b,\xi}{\min} \sum_{i=1}^{n} (1 - y_i(w^T\phi(x_i) + b))_+ + \frac{\lambda}{2} \| w \|^2$ under constraints $-\xi \leq 0, 1 - \xi_i - y_i(w^T\phi(x_i) + b) \leq 0, i = 1, \dots, n$
    - Problem of the type: $\underset{\theta}{\min} f(\theta)$ under constraints $g(\theta) \leq 0, i = 1, \dots, n$
        - Lagrangian: $J(\theta, \lambda) = f(\theta) + \lambda g(\theta), \lambda \geq 0$
        - There is a unique solution to $\underset{\theta}{\min}\,\underset{\lambda}{\max} J(\theta, \lambda)$
    - Let $\mathcal{L}(w, b, \alpha) = \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} \xi_i + \sum_i \alpha_i\, (1 - \xi_i - y_i(w^T\phi(x_i) + b)) - \sum_{i=1}^{n} \xi_i\, \mu_i, \forall i, \mu_i, \alpha_i \geq 0, \dots, n$
        - Karush-Kuhn-Tucker (KKT) conditions:
            - $\nabla_w \mathcal{L}(w, b, \xi, \alpha) = w - \sum_{i=1}^{n} \alpha_i\, y_i \phi(x_i) = 0$
            - $\nabla_b \mathcal{L}(w, b, \xi, \alpha) = -\sum_{i=1}^{n} \alpha_i\, y_i = 0$
            - $\nabla_\xi \mathcal{L}(w, b, \xi, \alpha) = C - \alpha_i - \mu_i = 0$
            - $\forall i, \alpha_i(1 - \xi_i - y_i(w^T\phi(x_i) + b)) = 0$
            - $\forall i, \mu_i \xi_i = 0$
- Dual: $\underset{\alpha}{\max} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i\, \alpha_j y_i y_j \phi(x_i)^T \phi(x_j)$ under constraints $0 \leq \alpha_i \leq C, \sum_{i=1}^{n} \alpha_i\, y_i = 0, i = 1, \dots, n$
    - For nonlinear feature map $\phi\colon \mathcal{X} \to \mathcal{F}$ we have nonlinear decision frontier
    - Solution: $f(x) = \operatorname{sign}(\sum_{i=1}^{n} \alpha_i\, y_i \phi(x_i)^T \phi(x) + b)$
- Kernel trick: $\phi(x_i)^T \phi(x)_j = k(x_i, x)$
    - Valid for positive definite symmetric kernels: every column vector $c \in \mathbb{R}^m, c^T K c = \sum_{i,j=1}^{m} c_i\, c_j k(x_i, x_j) \geq 0$
        - Any finite Gram matrix built from k and a finite number of elements of $\mathcal{X}$ is positive semi-definite
    - Polynomial of kernels is still a kernel
    - Examples
        - Trivial linear kernel: $k(x, x') = x^T x'$
        - Polynomial kernel: $k(x, x') = (x^T x' + c)^d$
        - Gaussian kernel: $k(x, x') = \exp(-\gamma \| x - x' \|^2)$ (universal approximator)
    - On complex (unstructured) or structured objects
- A training datapoint x_i is called a support vector if $\alpha_i^* \neq 0$.
    - A training data is a support vector if either it lies on or between the hyperplanes H_1 and H_{-1}
    - C is a hyperparameter that controls the compromise between the model complexity and the training classification error
- Support vector regression
    - ϵ-insensitive loss: $|y' - y|_\epsilon = \max(0, |y' - y| - \epsilon)$
    - Primal: $\underset{w,b,\xi}{\min} \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i^*)$ under constraints $-\xi \leq 0, -\xi^* \leq 0, y_i - (w^T\phi(x_i) + b) \leq \epsilon + \xi_i, (w^T\phi(x_i) + b) - y_i \leq \epsilon + \xi_i^*, i = 1, \dots, n$
    - Dual: $\underset{\alpha,\alpha^*}{\max} \sum_{i,j} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \phi(x_i)^T \phi(x_j) + \epsilon \sum_i (\alpha_i + \alpha_i^*) + \sum_i y_i\, (\alpha_i - \alpha_i^*)$ under constraints $\sum_i (\alpha_i - \alpha_i^*) = 0, 0 \leq \alpha_i \leq C, 0 \leq \alpha_i^* \leq C, i = 1, \dots, n$
        - Solution: $f(x) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) y_i \phi(x_i)^T \phi(x) + b$

Introduction to Deep Learning

- A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E
    - Supervised vs unsupervised
- Perceptron: $\hat{y}_i = f(x_i, w) = \operatorname{sign}(w^T x_i + w_0)$
    - Learning rule: if $y_i \neq \hat{y}_i$ update $w_{t+1} = w_t + \operatorname{sign}(y_i) x_i$
    - Converges if classes are linear separable
    - Convergence theorem: let $w^*, \| w^* \| = 1, \gamma > 0\colon \forall i \in [1, n], y_i x_i^T w^* \geq \gamma$
        - If $\exists R > 0, \| x_i \| \leq R$, the perceptron algorithm converges in at most $\frac{R^2}{\gamma^2}$ iterations
- Loss function: $l((x_i, y_i)_{i=1}^{n} | w) = \frac{1}{n} \sum_{i=1}^{n} l\,(f(x_i, w), y_i)$
    - Zero-one prediction loss: $1_{y \neq f(x,w)}$
    - Hinge loss: $(1 - yf(x, w))_+$
    - Squared difference: $(y - f(x, w))^2$
- Activation function:
    - Sigmoid function $\sigma(x) = \frac{1}{1 + \exp(-x)}$
        - $\frac{\partial \sigma}{\partial x} = \sigma(x)(1 - \sigma(x))$
    - Hyperbolic tangent: $\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$
    - Softmax: $o_i = softmax(h^{(l)})$ with $o_i = \frac{\exp(h^{(l),i})}{\sum_{k=1}^{K} \exp(h^{(l),k})}$
        - $\frac{\partial o_k}{\partial x_j} = \begin{cases} -o_j o_k & \text{if } k \neq j \\ -o_k(1 - o_k) & \text{if } k = j \end{cases}$
    - ReLU: $\max(0, x)$
- Probabilistic loss function: maximum likelihood estimation $L((x_i, y_i)_{i=1}^{n} | \theta) = \prod_{i=1}^{n} \hat{P}\,(y_i = 1 | x_i, \theta)^{y_i} \hat{P}(y_i = 0 | x_i, \theta)^{1-y_i} = \prod_{i=1}^{n} f\,(x_i, \theta)^{y_i} (1 - f(x_i, \theta))^{1-y_i}$
    - $\hat{\theta} = \underset{\theta}{\operatorname{argmax}} L((x_i, y_i)_{i=1}^{n} | \theta)$
    - Equivalently: minimize $l_{MLE}((x_i, y_i)_{i=1}^{n} | \theta) = -\log L((x_i, y_i)_{i=1}^{n} | \theta) = D_{KL}(\hat{p}_{data}, p_\theta) + C$
        - Sigmoid: $\frac{\partial l_{MLE}}{\partial w^j}((x_i, y_i)_{i=1}^{n} | \theta) = -\sum_{i=1}^{n} [\,y_i - f(x_i, w)] x_i^j$
        - Softmax: $l_{MLE}((x_i, y_i)_{i=1}^{n} | \theta) = -\sum_{i=1}^{n} \log \hat{o}_i^{y_i}$ or $l_{MLE}((x_i, y_i)_{i=1}^{n} | \theta) = -\sum_{i=1}^{n} \sum_{k=1}^{K} y_{i,k} \log \hat{o}_{i,k}$
- Gradient-based learning:
    - Given a learning rate ϵ: on a random sample set from the data, update $w_{t+1} = w_t - \epsilon \nabla_w l_{MLE}(w)$
    - Gradient on small batch of randomly selected points: stochastic gradient descent
- Multi-layer Perceptron: perceptrons can be combined as inputs to a new neuron
    - Fully connected layer: $h^{(l)} = \phi^{(l)}(w^{(l)T} x + b^{(l)})$
    - Optimization: binary case with softmax: $l_{MLE}(\theta) = -\sum_{i=1}^{n} \log \left(\frac{\exp(h^{(l)}, y_i)}{\sum_{k=1}^{K} \exp(h^{(l),k})} \right)$
    - Backpropagation
    - With one hidden layer of $p + 1$ inputs is dense from a compact subset of $\mathbb{R}^p$ to $\mathbb{R}$: there exist a number of neurons N such that the MLP is arbitrarily close to a function f in that space
    - Evaluate for overfitting and underfitting: split data into train, validation and test
    - Learning rate: initialize largely and iteratively decay it
    - Regularization: weight decay: tradeoff between fitting the training data and keeping the parameters small $l_{Reg}(w) = l_{MLE}(w) + \lambda w^T w$
        - Update rule $w_{t+1} = (1 - 2\lambda) w_t - \epsilon \nabla_w l_{MLE}(w)$
    - Weights initialization: highly experimental
        - Unsupervised pre-training: fine-tuning denoising autoencoder
- Vanishing/saturated gradients: activation functions with null derivatives
- Dropout: randomly remove parts of the hidden layers to refrain the neural network from copying data
- Early stopping: fall back the model to best epoch
- Adaptative gradient: adjust learning rate during training
- Autoencoders: feed-forward neural network trained to rebuild its input through a hidden representation (unsupervised)
    - Loss function: sum of squared differences: $l(x|\theta) \frac{1}{2} \sum_k (\hat{x}^k - x^k)^2$
        - Assumes data coming from an isotropic Gaussian
        - Adapt the loss to new data:
            - Assume the data is from an isotropic gaussian with mean μ: $p(x|\mu) = \frac{1}{(2\pi)^{\frac{D}{2}}} \exp\left(-\frac{1}{2} \sum_k (x^k - \mu^k)^2\right)$
            - Choose $\mu = W^T h + b$ and loss $l(x) = -\log p(x|\mu)$
    - The linear encoder is actually optimal for minimizing the training square error (if the data is standardized, the encoder corresponds to the PCA)
    - Denoising autoencoder: add noise to induce the autoencoder to learn about the structure of the data
- Deep learning: especially relevant for complex inputs, better data representation (distributed representations allows for great generalization, discovers explanatory factors through the training, exponential gain through the depth)

Generative models

- Linear Discriminant Analysis (LDA)
    - Assumption: Each class follows a normal distribution with different mean but same covariance $(x \sim N(\mu_k, \Sigma))$, so we want to maximize: $P(y = k|x) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma^{-1} (x - \mu_k)\right)$

$$\begin{aligned} \hat{y} &= \underset{k}{\operatorname{argmax}}[\ln(\pi_k) + \ln(P_k(x))] \\ &= \underset{k}{\operatorname{argmax}}\left[\ln(\pi_k) + x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k\right] \\ &= \underset{k}{\operatorname{argmax}}[\delta_k(x)] \end{aligned}$$

    - $\delta_k(x)$ called discriminant score. Boundary between classes k_1, k_2 when $\delta_k(x) = \delta_k(x)$ (which can be rewritten as $Ax + b = 0$ so it's linear).
    - Supervised learning: μ_k is the average for the data points we have for each class, Σ is the covariance of all the training dataset.
- Quadratic Discriminant Analysis (QDA)
    - Assumption: Each class follows a normal with different mean and covariance $(x \sim N(\mu_k, \Sigma_k))$.

$$\hat{y} = \underset{k}{\operatorname{argmax}}\left[\ln(\pi_k) - \frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) - \frac{1}{2} \ln|\Sigma_k|\right]$$

    - It's the same as LDA but since we assume different covariances we end up with a quadratic boundary between classes.
- Gaussian Mixture Models (GMM)
    - Mixture models assume the data has been generated from a linear superposition of different distributions $p(x) = \sum_{k=1}^{K} \pi_k\, \mathcal{N}(x|\mu_k, \Sigma_k)$. GMMs are a particular case in which we assume all of these distributions are Gaussian.

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma\,(z_{ik}) x_i$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma\,(z_{ik})(x_i - \mu_k)(x_i - \mu_k)^T$$

$$\gamma(z_{ik}) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}$$

    - Unsupervised learning.
    - No closed-form solution: We don't have $\gamma(z_{ik})$.
    - Not identifiable.
    - Estimated with Expectation Maximization algorithm.
        - Initialize μ_k, Σ_k and π_k (ex: With K-means).
        - E-step: Evaluate $\gamma(z_{ik})$ using current estimates of μ_k, Σ_k and π_k.
        - M-step: Estimate μ_k, Σ_k and π_k using the computed $\gamma(z_{ik})$.
        - Slower and more complex than K-means.
        - Possible multiple local maxima → re-run with different initializations.
        - Possible problem with singularities ($\mu_k = x_i$ and $\Sigma_k \to 0$) → randomly reset μ_k and Σ_k.