# Machine Learning 1

## 1 Linear Regression

- Supervised Learning (regression, classification)
- Unsupervised Learning (clustering, dimensionality reduction)
- Matrix: single samples are rows
- Derivative of vector input function is column vector
- $\boldsymbol{\nabla_x A x} = \boldsymbol{A}^\top$, $\boldsymbol{\nabla_x x}^\top \boldsymbol{x} = 2\boldsymbol{x}$, $\boldsymbol{\nabla_x x}^\top \boldsymbol{A x} = 2\boldsymbol{A x}$
- Linear Regression: fit line $y = f(x) + \varepsilon = w_0 + w_1 x + \varepsilon$ (Guassian noise $\varepsilon \sim N(0, 1)$)
- Minimize summed/mean squared error SSE $= \sum_{i=1}^{N} (y_i - f(\boldsymbol{x}_i))^2$ (differentiable, easy to optimize, estimates mean of target function)
- Multiple inputs: SSE $= (\boldsymbol{y} - \boldsymbol{X w})^\top (\boldsymbol{y} - \boldsymbol{X w})$ with $\boldsymbol{X} = \begin{bmatrix} 1 & \boldsymbol{x}_1^\top \\ \vdots & \vdots \\ 1 & \boldsymbol{x}_n^\top \end{bmatrix}$
- Least squares solution ($\boldsymbol{\nabla_w}$SSE $= 0$): $\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$ - closed form because SSE convex for linear $f(\boldsymbol{x})$ (one minimum) and quadratic in $w$ (easy to obtain)
- $R^2 = 1 - \frac{\sum (\hat{y}_n - y_n)^2}{\sum (y_n - \bar{y})^2}$ (quality: how much variation in $y$ explained by variation in $x$)
- Generalized: $f(\boldsymbol{x}) = \tilde{\boldsymbol{x}}^\top \boldsymbol{w} \rightarrow f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{w}$, still linear in $w$ ($\phi_i$: basis functions)
- $\boldsymbol{w^*} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y}$ with $\boldsymbol{\Phi} = \begin{bmatrix} \phi_1^\top \\ \vdots \\ \phi_n^\top \end{bmatrix}$ (learn any function with suitable $\phi_i$)
- Overfitting: model too complex, fits noise / Underfitting: model too simple for data
- Regularization (limit model): Regularization term in cost function with factor $\lambda$
- $L_{\text{ridge}} = (\boldsymbol{y} - \boldsymbol{\Phi w})^\top (\boldsymbol{y} - \boldsymbol{\Phi w}) + \lambda \boldsymbol{w}^\top \boldsymbol{w}$ (weight decay / ridge regression)
- $\boldsymbol{w}^*_{\text{ridge}} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y}$ (easier to invert due to full rank)

## 2 Linear Classification

- Expectation of function wrt a distribution: $\mathbb{E}_p[f(x)] = \int p(x)f(x)dx$
- Conditional expectation: $\mathbb{E}_p[f(x)|Y = y] = \int p(x|y)f(x)dx$
- Chain rule: $\mathbb{E}_p[f(x)] = \int p(y)\mathbb{E}_p[f(x)|Y = y]dx$
- Monte-carlo: estimate expectation by samples
- Covariance: $\boldsymbol{\Sigma} = \mathbb{E}_p[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top]$, diagonal: variability, other: correlation
- Bernoulli distribution: $p(x) = \mu^x(1 - \mu)^{(1-x)}$ (coin toss)
- Multinomial / Categorical Distribution: $p(c) = \Pi\, \mu_k^{\boldsymbol{h}_{c,k}}$ with 1-hot-encoding (die)
- Gaussian Distribution: $p(x) = N(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{(x-\mu)^2}{2\sigma^2}\}$
- Multivariate: $p(\boldsymbol{x}) = N(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{|2\pi\boldsymbol{\Sigma}|}} \exp\{-\frac{(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}{2}\}$
- Maximum Likelihood Estimation: $\boldsymbol{\theta}_{\text{ML}} = \text{argmax}_\theta \text{loglik}(\boldsymbol{\theta}, D)$
- Linear Gaussian model $p_\theta(y|\boldsymbol{x}) = N(y|\boldsymbol{w}^\top \tilde{\boldsymbol{x}}, \sigma^2) \rightarrow$ MLE solution equvialent to least squares, but variance can also be obtained
- Generative model: Assume form of $p(c)$, $p(x|c)$, learn them, predict: compute $p(c|x) \rightarrow$ learn full joint distribution of data (hard), gaussian assumption $\rightarrow$ error
- Discriminative Model: Assume form of $p(c|x)$ and estimate parameters directly from data (simpler than generative modelling, only considers points on border)
- Linear classifier: $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ ($\boldsymbol{w}$ normal to line, $b$ is bias)
- Counting number of misclassifcations as loss is very difficult to optimize (NP-hard)
- Regression loss is not robust to outliers (labels restricted to 0, 1)
- Solution: squash output with sigmoid (bounded between 0 and 1)
- Probabilistic View: $p(c|\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)^c(1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b))^{1-c} \rightarrow$ optimize loglikelihood (logistic regression): cross-entropy loss: $-\sum_i c_i \log f(\boldsymbol{x}_i) + (1 - c_i)\log(1 - f(\boldsymbol{x}_i)) \rightarrow$ convex but no closed form $\rightarrow$ gradient descent
- Generalized: Use basis functions to make data linear seperable in feature space
- L2 regularization loss: $\text{penalty}(\tilde{\boldsymbol{w}}) = ||\tilde{\boldsymbol{w}}||^2$
- General optimization form: $\text{argmin}_\theta \sum l(\boldsymbol{x}_i, \boldsymbol{\theta}) + \lambda\, \text{penalty}(\boldsymbol{\theta})$
- Gradient descent: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t)$ with learning rate $\eta$

- Terminate: change small, gradient small, change in value small, or after fixed time
- Stochastic: use one sample for step (good far away, struggle to find exact optimum)
- Stochastic approximation theory: SGD converges to optimum for strictly convex functions if $\sum \eta_t = \infty$ and $\sum \eta_t^2 < \infty$ (for example $\eta_t = \frac{1}{t}$)
- Stochastic gradients often better than batch since data-set contains redundancy
- Mini-Batches: intermediate between stochastic and batch, preferable for GPU
- Softmax: $p(c = i|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_i^\top \boldsymbol{\phi}(\boldsymbol{x}))}{\sum \exp(\boldsymbol{w}_k^\top \boldsymbol{\phi}(\boldsymbol{x}))}$ (each class gets a weight vector)
- Multiclass Classifcation: $p(c|\boldsymbol{x}) = \Pi\ p(c = k|\boldsymbol{x})^{\boldsymbol{h}_{c,k}}$ (use softmax) $\rightarrow$ can be optimized by gradient descent

# 3 Model Selection

- Model complexity (linear regression: features, $\lambda$; decision tree: depth, number of leaves; NNs: layers, neurons; SVMs: features, regularization; Gaussian Processes: kernel bandwith) $\rightarrow$ model selection problem
- True risk (unknown) vs. emperical risk (can be evaluated)
- Overfitting (small emperical/high true risk)/Underfitting (high emperical/true risk)
- Expected loss for model and data set size $n$: $R(\hat{f}_{D_n}) = \mathbb{E}_{D_n}[\mathbb{E}_{x,y}[(\hat{f}_{D_n}(\boldsymbol{x}) - y)^2)]]$
  $= \mathbb{E}_{D_n}[\mathbb{E}_x[(\hat{f}_{D_n}(\boldsymbol{x}) - \hat{f}_*(\boldsymbol{x}))^2]] + \mathbb{E}_x[(\hat{f}_*(\boldsymbol{x}) - f(\boldsymbol{x}))^2] + \sigma^2$
  $=$ Variance + Bias$^2$ + Noise (bias (structure error) due to restriction of model, variance due to randomness of data set, $\hat{f}_{D_n}$: estimate of $f$ from data $D_n$, $\hat{f}_*(\boldsymbol{x}) = \mathbb{E}_{D_n}[\hat{f}_{D_n}(\boldsymbol{x})]$: best model possible, $y = f(\boldsymbol{x}) + \varepsilon$)
- Underfitting: low variance / high bias; Overfitting: high variance / low bias
- Hold-out method: Judge generalization error with validation data set to pick model (needs more data, unlucky splits can give misleading results)
- Cross validation: Split dataset into $k$ folds, use each as validation once
- Leave-One-Out: $k = n$ / Random sub-sampling: Random points used in each fold
- Avoid overfitting: low complexity, regularize, early stopping, noise, augmentation
- Regularization: penalty$_{L2}(\theta) = ||\theta||_2$ (optimizing easy)/penalty$_{L1}(\theta) = ||\theta||_1$ (hard, leads to sparse solutions)
- Early stopping: similar effects to L2, efficient (only store best and current weights), simple, no hyper parameter (but needs validation data)
- Linear regression: input noise (leads to more robust solutions) is the same as L2

# 4 Nearest Neighbours, Trees and Forests

- Non-parametric methods: use training data directly for prediction (complexity adapts to training data, very fast training, slow predictions, hard for high dimensions)
- KNN: needs lots of training data and less than 20 attributes, can learn complex functions, regression works similar
- Increasing k reduces variance, increases bias
- Euclidean distance when each variable has same unit, otherwise normalize data
- Cosine Distance (documents, images), Hamming Distance (string data/categorical features), Manhatten Distance (coordinate-wise), Mahalanobis Distance (unaffected by coordinate transformations)
- Performance of KNN degrades with irrelevant dimensions/high dimensions (most points far away) $\rightarrow$ dimensionality reduction, feature selection
- KD-Tree to find neighbours; Build: choose dimension by longest hyperrectangle side, median as pivot; Traverse: move down tree, find region containing $\boldsymbol{x}$, find closest $\boldsymbol{x}^*$, move up for regions intersecting hypersphere, update $\boldsymbol{x}^*$
- Regression/Classification Tree: split data into two at each node using criterion
- Splitting criterion regression: Minimum residual sum of squares: RSS $= \sum_{\text{left}}(y_i - \overline{y}_L)^2 + \sum_{\text{right}}(y_i - \overline{y}_R)^2$, $\overline{y}$: average label subtree (variance in subtrees minimized)
- Criterion classification: Minimum entropy: score $= N_L H(p_L) + N_R H(p_R)$, $H(p_i) = -\sum_k p_i(k) \log p_i(k)$: subtree entropy, $p_i(k)$: proportion of class $k$ in subtree $i$
- Stop: Minimum number of samples per node / maximum depth (tree complexity)
- Trees: easy to compute, no distributional assumption, non-linear, automatic variable selection, easy to interpret, lower accuracy, sensitive to data change
- Random Forests: use multiple trees to improve accuracy
- Bagging: Fit trees to bootstrap samples from data (combine by voting/averaging)
- Ideal: linear variance reduction (trees correlated $\rightarrow$ reduction still significant)
- Bagging: less variance, bias unaffected $\rightarrow$ use strong trees (high variance/low bias)
- Random Forests: Also randomize considered variables at each splitting criterion, grow to maximum depth (loss of interpretability, good accuracy, less unstable)

# 5 Dimensionality Reduction and Clustering

- Motivation: Invert $\boldsymbol{X}^\top \boldsymbol{X}$ for linear regression: $d \times d \rightarrow O(d^3) \rightarrow$ find $d_{\text{new}} \ll d$
- Find (linear) mapping $\boldsymbol{x}_i \rightarrow \boldsymbol{z}_i$ to lower dimension with $\boldsymbol{z}_i = \boldsymbol{W}\boldsymbol{x}_i$
- Orthonormal basis system: $\boldsymbol{x} = \sum_i^D z_i \boldsymbol{u}_i \rightarrow z_i = \boldsymbol{u}_i^\top \boldsymbol{x} \rightarrow$ only use subset for dimensionality reduction (minimize squared reproduction error $\sum ||\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i||^2$)
- Minimizing error $\Leftrightarrow$ maximizing variance of projection (with zero mean data)
- Principle component analysis: find principal directions $\boldsymbol{u}_i$ and their variance $\lambda_i$
- $\boldsymbol{u}_1 = \text{argmax}_u \frac{1}{N} \sum (\boldsymbol{u}^\top (\boldsymbol{x}_i - \boldsymbol{\mu}))^2$ s.t. $\boldsymbol{u}^\top \boldsymbol{u} = 1$, $\boldsymbol{u}_2$ maximizes variance in orthogonal complement of $\boldsymbol{u}_1$
- Objective can be written in terms of sample covariance: $E(\boldsymbol{u}) = \boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u}$
- Constraint Optimization: Lagrangian Multipliers ($L$ = objective $-$ multiplier $\cdot$ constraint): $\min_x x^2$ s.t. $x \geq b \rightarrow \min_x \max_\alpha L(x, \alpha) = x^2 - \alpha(x - b)$ s.t. $\alpha \geq 0$ (Min forces max to behave such that constraints are satisfied)
- Dual formulation: $\boldsymbol{\lambda}^* = \text{argmax } g(\boldsymbol{\lambda}),\ g(\boldsymbol{\lambda}) = \min_x L(x, \boldsymbol{\lambda})$ s.t. $\lambda_i \geq 0, \boldsymbol{x}^* = \text{argmin}_x\ L(\boldsymbol{x}, \boldsymbol{\lambda}^*)$ (swap min/max)
- Slaters condition: convex objective/constraints $\Rightarrow$ dual $\Leftrightarrow$ primal (original)
- PCA: $\boldsymbol{u}_1 = \text{argmax}_u\ \boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u}$ s.t. $\boldsymbol{u}^\top \boldsymbol{u} = 1 \Rightarrow L(\boldsymbol{u}, \lambda) = \boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u} + \lambda(\boldsymbol{u}^\top \boldsymbol{u} - 1) \Rightarrow \boldsymbol{\Sigma} \boldsymbol{u} = \lambda \boldsymbol{u}$ (eigenvalue problem, largest value: maximum variance, vector: direction)
- Representation has minimum MSE of all linear representations of same dimension
- PCA: Subtract mean, (normalize variance of each dimension), choose first $M$ largest eigenvalues/their vectors of $\boldsymbol{\Sigma}$, $\boldsymbol{z}_i = \boldsymbol{B}^\top (\boldsymbol{x}_i - \boldsymbol{\mu})$, reprojection: $\tilde{\boldsymbol{x}}_i = \boldsymbol{\mu} + \boldsymbol{B}\boldsymbol{z}_i$
- Choose $M$: based on application performance/based on captured variance
- Applications: face detection, morphing, natural image patches, ...
- Clustering Group data using similarity measure ($D(A, B) = D(B, A),\ D(A, B) = 0 \Leftrightarrow A = B,\ D(A, B) \leq D(A, C) + D(B, C)$)
- Hierarchical Clustering: Dendrogram (similarity: height of lowest shared node)
- Outlier: single isolated branch
- Heurestic search of all possible trees: Bottom-up / Top-down (find best division)
- Bottom-up: each sample in own cluster, merge closest two clusters, until single cluster left (requires distance measure for samples and clusters)

- Cluster similarity: single linkage (minimum distance between two points) / complete linkage (maximum distance) / average linkage / centroid linkage
- Hierarchical: any number of clusters, $O(n^2)$, local optima, subjective interpretation
- Flat Clustering: K-Means: minimize quantization error (sum of squared distances) $\text{SSD}(C, D) = \sum d(\boldsymbol{x}_i, c(\boldsymbol{x}_i))^2$
- Iteration: 1. pick $K$ random centroids $c_i$, 2. assign each point to closest $c_i$, 3. move centroids to mean of assigned points, 4. go to step 2 until no change
- SSD $= \sum_i \sum_k \delta_{ik} d(\boldsymbol{x}_i, \boldsymbol{c}_k)^2 \rightarrow$ assigment minimizes w.r.t. $\delta_{ik}$, adjustment w.r.t. $\boldsymbol{c}_k$
- K-Means locally minimizes SSD (depends on intialization, global NP-hard)
- K-Means++: first centroid random, each following centroid furthest from all others
- Choose $K$: objective function decrease on holdout set or Knee-finding method
- Knee-finding: plot SSD for $K$, pick point where decrease is no longer steep
- K-Means: converges quickly, local optima, not applicable to categorical data/noisy data/outliers, clusters must be convex

# 6 Density Estimation and Expectation Maximization

- Non-parametric models (don't know form of class-conditional density) $\rightarrow$ estimate directly from data (histograms, kernel density, KNN)
- Histograms: general, need exponential data (curse of dimensionality), fixed region size, $p(\boldsymbol{x}) \approx \frac{K}{NV}$ ($K$ points in region $R$, $N$: total points, $V$: volume of $R$)
- Center $R$ on $\boldsymbol{x}$: Kernel density: fix $V$, determine $K$, KNN: fix $K$, determine $V$
- Kernel Density Estimation: $k(\boldsymbol{x}, \boldsymbol{y})$: non-negative, distance-dependent: $k(\boldsymbol{x}, \boldsymbol{y}) = g(\boldsymbol{x} - \boldsymbol{y})$, $V = \int g(\boldsymbol{u}) d\boldsymbol{u}$, $K(\boldsymbol{x}_*) = \sum g(\boldsymbol{x}_* - \boldsymbol{x}_i) \rightarrow p(\boldsymbol{x}_*) \approx \frac{K(\boldsymbol{x}_*)}{NV}$
- Parzen Window (hypercubes): $g(\boldsymbol{u}) = 1$ if $|u_j| \leq \frac{h}{2}, j = 1 \ldots d,$ else $0 \Rightarrow p(\boldsymbol{x}_*) \approx \frac{K(\boldsymbol{x}_*)}{Nh^d}$, $h$: bandwidth, $d$: dimensionality (easy to compute, not very smooth)
- Gaussian Kernel: $g(\boldsymbol{u}) = \exp(-\frac{||\boldsymbol{u}||^2}{2h}) \rightarrow p(\boldsymbol{x}_*) \approx \frac{1}{N\sqrt{(2\pi h)^d}} \sum \exp(-\frac{||\boldsymbol{x}_* - \boldsymbol{x}_i||^2}{2h})$ (smooth, infinite support, computationaly intensive, bigger $h \rightarrow$ smoother curve)
- Cross-validation for bin size/bandwidth/neighbours (highest likelihood on test-set)
- Mixture Models generality of non-parametric models and efficiency of parametric models $\rightarrow$ create complex distribution by combining simple ones (e.g. Gaussians)
- Mixture coefficient $\cdot$ component: $p(\boldsymbol{x}) = \sum p(k) p(\boldsymbol{x}|k) \sim$ any smooth density

- $p(k) = \pi_k \geq 0, \sum \pi_k = 1, p(\boldsymbol{x}|k) = N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \boldsymbol{\theta} = \{\pi_1, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \ldots, \pi_K, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K\}$
- Gradient descent on marginal log-likelihood? $\rightarrow$ possible, inefficient (depends on all components, no closed form, slow convergence, sum does not go well with log)
- Mixture models $\rightarrow$ latent variable models (observed variables $\boldsymbol{x}$ and latent variables $\boldsymbol{z}$): $p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ (parametric model), $p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_z p(\boldsymbol{x}, z|\boldsymbol{\theta})$ (marginal distribution)
- Kullback-Leibler Divergence (similarity of distributions): $\mathrm{KL}(q||p) = \sum_x q(\boldsymbol{x}) \log \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})}$ (non-negative, zero for same distribution, non-symmetric $\rightarrow$ no distance metric)
- Expectation-Maximization algorithm estimates latent variable models (iteratively increases lower bound of the marginal log-likelihood) $\rightarrow$ local optima
- $\log p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_z q(z) \log \frac{p(\boldsymbol{x}, z|\boldsymbol{\theta})}{q(z)} + \sum_z q(z) \log \frac{q(z)}{p(z|\boldsymbol{x})} = L(q, \boldsymbol{\theta}) + \mathrm{KL}(q(z)||p(z|\boldsymbol{x}))$ (decomposition holds for any $q(z)$, it makes optimization much simpler)
- $L(q, \boldsymbol{\theta}) \leq \log p(\boldsymbol{x}|\boldsymbol{\theta})$ contains joint distribution $\rightarrow$ easier to optimize (often convex)
- Expectation step: Find $q(z)$ to minimize KL $\rightarrow q(z) = p(z|\boldsymbol{x}, \boldsymbol{\theta}_{\text{old}}) = \frac{p(\boldsymbol{x}, z|\boldsymbol{\theta}_{\text{old}})}{\sum_z p(\boldsymbol{x}, z|\boldsymbol{\theta}_{\text{old}})}$ (closed form for discrete $z$) $\Rightarrow \mathrm{KL} = 0 \Rightarrow$ lower bound $L(q, \boldsymbol{\theta}_{\text{old}})$ tight at $\boldsymbol{\theta}_{\text{old}}$
- Maximization step: Maximize $L(q, \boldsymbol{\theta})$: $\boldsymbol{\theta}_{\text{new}} = \operatorname{argmax} \sum_z q(z) \log p(\boldsymbol{x}, z|\boldsymbol{\theta}) + \text{const}$
- Full dataset: $L(q, \boldsymbol{\theta}) = \sum_i (\sum_k q_{ik} \log p(\boldsymbol{x}_i, k|\boldsymbol{\theta}) - \sum_k q_{ik} \log q_{ik})$ with $q_{ik} = q_i(z = k)$ (one latent variable per data-point)
- Gaussian mixture models E-step: Compute "responsibilities" of components: $q_{ik} = \frac{\pi_k N(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j N(\boldsymbol{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} = p(z = k|\boldsymbol{x}_i)$
- M-step: Seperate updates for additive objectives: $\boldsymbol{\pi} = \operatorname{argmax} \sum_i \sum_k q_{ik} \log \pi_k$, $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k = \operatorname{argmax} \sum_i q_{ik} \log N(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \rightarrow$ weighted ML estimation
- $\pi_k = \frac{\sum_i q_{ik}}{N}$, $\boldsymbol{\mu}_k = \frac{\sum_i q_{ik} \boldsymbol{x}_i}{\sum_i q_{ik}}$, $\boldsymbol{\Sigma}_k = \frac{\sum_i q_{ik} (\boldsymbol{x}_i - \boldsymbol{\mu}_k)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^\top}{\sum_i q_{ik}}$
- EM for GMMs: Initialize (K-Means for component means and fixed covariance), until convergence: E-step (responsibilities $q_{ik}$), M-step (update $\boldsymbol{\pi}$, $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$)
- EM very sensitive to intialization, K-Means special case of EM
- More components $\rightarrow$ better likelihood (beware of overfitting)
- EM for dimensionality reduction (probabilistic PCA): $\boldsymbol{x} = \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\varepsilon}$, (latent variable $\boldsymbol{z}$: low dimensional representation, $\boldsymbol{\mu}$: constant offset, $\boldsymbol{\varepsilon} \sim N(0, \sigma^2 \boldsymbol{I})$: noise)
- Continuous latent variable: $p(\boldsymbol{z}) = N(\boldsymbol{0}, \boldsymbol{I})$, observation model $p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta}) = N(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$ with parameters $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{\mu}, \sigma^2\}$
- Generative process: sample $\boldsymbol{z} \sim N(\boldsymbol{0}, \boldsymbol{I})$, project: $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}$, add noise: $\boldsymbol{x} = \boldsymbol{y} + \boldsymbol{\varepsilon}$

- Maximize marginal loglike($\boldsymbol{\theta}$) $= \sum_i \log(\int_z N(\boldsymbol{x}|\boldsymbol{W}\boldsymbol{z}+\boldsymbol{\mu}, \sigma^2\boldsymbol{I})N(\boldsymbol{z}|\boldsymbol{0}, \boldsymbol{I})d\boldsymbol{z}) \rightarrow$ EM
- E-step: Posterior $q_i(\boldsymbol{z}) = p(\boldsymbol{z}|\boldsymbol{x}_i, \boldsymbol{\theta})$ with $\boldsymbol{\mu}_{z|x_i} = (\boldsymbol{W}^\top\boldsymbol{W} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{W}^\top(\boldsymbol{x}_i - \boldsymbol{\mu}), \boldsymbol{\Sigma}_{z|x_i} = \sigma^2(\boldsymbol{W}^\top\boldsymbol{W} + \sigma^2\boldsymbol{I})^{-1}$ (only possible because $\boldsymbol{x}$ is linear in $\boldsymbol{z}$)
- M-step: $L(q, \boldsymbol{\theta}) = \sum_i \mathbb{E}_{q_i(z)}[\log p(\boldsymbol{x}_i|\boldsymbol{z}, \boldsymbol{\theta})] + \text{const} \approx \sum_i \log p(\boldsymbol{x}_i|\boldsymbol{z}_i, \boldsymbol{\theta})$ with $\boldsymbol{z}_i \sim q_i(\boldsymbol{z})$ (approximate with single sample per $\boldsymbol{x}_i$) $\rightarrow$ solution: standard least squares: $\begin{bmatrix}\boldsymbol{\mu} \\ \boldsymbol{W}\end{bmatrix} = (\boldsymbol{Z}^\top\boldsymbol{Z})^{-1}\boldsymbol{Z}^\top\boldsymbol{X}, \sigma^2 = \frac{1}{nd}\sum_i^n\sum_k^d(y_{ik} - x_{ik})^2$
- PCA with eigenvectors preferred (one step $\rightarrow$ very fast), probabilistic PCA provides density, helps understand EM and more complex dimensionality reduction methods
- EM: assumes KL can be zero (posterior can be evaluated analytically), $\boldsymbol{z}$ must be discrete/linear gaussian $\rightarrow$ Variational Bayes/Inference can work with KL $> 0$

# 7 Kernel Methods

- Kernel: represent $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ by $[\boldsymbol{K}]_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ ($k : X \times X \rightarrow \mathbb{R}$: comparison)
- Modularity between choice of $k$ and algorithm, poor scalability
- Positive definite kernel function $k$: symmetric, $\boldsymbol{K}$ is always positive definite
- $k(\boldsymbol{x}, \boldsymbol{x}') = \langle\phi(\boldsymbol{x}), \phi(\boldsymbol{x}')\rangle$: positive definite kernel (arbitrary feature function $\phi$)
- Theorem: positive definite (p.d.) kernel $\Leftrightarrow$ associated feature space
- Kernel for polynomial features of degree $d$: $k(\boldsymbol{x}, \boldsymbol{x}') = \langle\boldsymbol{x}, \boldsymbol{x}'\rangle^d$
- Gaussian kernel: $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\frac{||\boldsymbol{x}-\boldsymbol{y}||^2}{2\sigma^2})$ with bandwidth $\sigma$ (most used kernel)
- Gaussian kernel is inner product of two infinite dimensional feature vectors $\rightarrow$ p.d.
- Kernel trick: feature based algorithms can use infinite dimensional feature space if rewritten to contain inner products of feature vectors $\rightarrow$ better than linear features
- Kernel ridge regression: $\boldsymbol{w}^* = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y} = \boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \lambda\boldsymbol{I})^{-1}\boldsymbol{y} \rightarrow d \times d$ matrix inversion (infinite) to $N \times N$ matrix inversion ($\boldsymbol{K}$, by using matrix identity)
- $\boldsymbol{w}^*$ still $d$-dimensional, but can evaluate $f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top\boldsymbol{w}^* = \boldsymbol{k}(\boldsymbol{x})^\top(\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y} = \sum_i \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x})$ with $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}$
- Comparison to linear regression with gaussian features: kernel allows setting centers adaptively (fixed without kernel trick)
- Choose hyper-parameter bandwidth via cross-validation
- Kernel methods (ridge regression, gaussian processes, SVMs): must store all samples, high computation, flexible representation, good for small data, hard to scale

# 8 Support Vector Machines

- Classification: class labels 1 and -1 for SVMs ($f(\boldsymbol{x}_i)y_i > 0$)
- Scalar projection of $\boldsymbol{a}$ on $\boldsymbol{b}$: $a_b = \frac{\boldsymbol{a}^\top \boldsymbol{b}}{||\boldsymbol{b}||}$
- Support vectors: data points closest to decision boundary (other samples ignored), maximize margin $\rho$
- Maximum margin classifier has smaller complexity $\Rightarrow$ generalizes better
- Distance between point $\boldsymbol{x}_i$ and line: $r = \frac{\boldsymbol{w}^\top \boldsymbol{x}_i + b}{||\boldsymbol{w}||}$
- Choose scaling for $\boldsymbol{w}, b$ so that $\boldsymbol{w}^\top \boldsymbol{x}_+ + b = 1$ (positive support vector), $\boldsymbol{w}^\top \boldsymbol{x}_- + b = -1$ (negative support vector) $\rightarrow \rho = \frac{2}{||\boldsymbol{w}||}$
- Optimization problem: $\text{argmax}_w \frac{2}{||\boldsymbol{w}||}$ s.t. $\boldsymbol{w}^\top \boldsymbol{x}_i + b \geq 1$ if $y_i = 1, \leq -1$ if $y_i = -1$ (one positive + negative point satisfy equality, otherwise weight could be reduced)
- Reformulation: $\text{argmin}_w ||\boldsymbol{w}||^2$, s.t. $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1$ (convex, single optimum)
- Choose trade-off between margin and accuracy (for outliers) $\rightarrow$ slack-variables $\xi_i \geq 0$ allow violation of margin: $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i \Rightarrow \text{argmin}_{w,\xi} ||w||^2 + C \sum \xi_i$ ($C$: inverse regularization, small $\rightarrow$ large $\rho$, large $\rightarrow$ small $\rho$, infinite $\rightarrow$ hard $\rho$)
- Reformulated as unconstrained optimization: $\text{argmin}_w \ ||\boldsymbol{w}||^2 + C \sum_i \max(0, 1 - y_i f(\boldsymbol{x}_i)) = \text{regularization} + \text{hinge loss} \rightarrow$ convex, one minimum, but not differentiable $\rightarrow$ similar to logistic regression loss
- Hinge loss: $\max\{0, 1 - y_i f(\boldsymbol{x}_i)\}$ / Logistic loss: $\log(1 + \exp(-y_i f(\boldsymbol{x}_i))) \rightarrow y_i f(\boldsymbol{x}_i)$ should be large for both / saturates if it gets too large
- Sub-gradient: Any $\boldsymbol{g}$ at point $\boldsymbol{x}$ so that $f(\boldsymbol{z}) \geq f(\boldsymbol{x}) + \boldsymbol{g}^\top (\boldsymbol{z} - \boldsymbol{x})$, if $f$ is differentiable at $\boldsymbol{x} \Rightarrow \boldsymbol{g} = \nabla f(\boldsymbol{x})$
- Let $f(x) = \max\{f_1(x), f_2(x)\}$. $f_1(x) = f_2(x) \Rightarrow g \in [\nabla f_1(x), \nabla f_2(x)]$
- Sub-gradient descent: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \eta \boldsymbol{g}$ (does not always decrease $f$, store best $\boldsymbol{x}^*$) SVMs: each iteration, pick random $(\boldsymbol{x}_i, y_i)$. $y_i f(\boldsymbol{x}_i) < 1 : \boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta(2\boldsymbol{w}_t - C y_i \boldsymbol{x}_i)$, otherwise $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta 2 \boldsymbol{w}_t$
- SVM: classification standard in 90s/00s (pedestrian detection, text categorization, character recognition, bioinformatics), extends to regression, outperformed by NNs
- Kernel SVM: Dual derivation: $\boldsymbol{w}^* = \sum_i \lambda_i y_i \phi(\boldsymbol{x}_i)$ ($\lambda_i$: constraint coefficient)
- $\frac{\partial L}{\partial b} = -\sum_i \lambda_i y_i \Rightarrow \sum_i \lambda_i y_i = 0$ no solution for $b$, but additional condition ($b$ can be computed from $\boldsymbol{w}$: $b = y_i - \boldsymbol{w}^\top \phi(\boldsymbol{x}_i)$ (for $\boldsymbol{x}_i$ on margin))

- Kernel trick for SVMs: $g(\boldsymbol{\lambda}) = \sum \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_j)$
- Dual optimization (slack variables): $\max_\lambda \sum \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ s.t. $C \geq \lambda_i \geq 0$, $\sum \lambda_i y_i = 0$ with $b = y_k - \sum_i y_i \lambda_i k(\boldsymbol{x}_i, \boldsymbol{x}_k)$ where $C > \lambda_k > 0$ and $f(\boldsymbol{x}) = \sum_i y_i \lambda_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b$ (upper bound $C$ limits $\lambda_i$ so misclassifcations allowed)
- Control overfitting: set $C$ (low $C \rightarrow$ low complexity), choose kernel, vary bandwidth

# 9 Bayesian Machine Learning

- Estimate $\boldsymbol{\theta}^*$ uncertainty, infinite predictors (mean) $\rightarrow$ give prediction uncertainty
- Compute posterior $p(\boldsymbol{\theta}|D) = \frac{p(D|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(D)}$, $p(D|\boldsymbol{\theta})$: data likelihood, $p(\boldsymbol{\theta})$: prior (subjective belief), $p(D)$: evidence (normalization, later used for model comparison)
- Compute predictive distribution (marginal likelihood) $p(\boldsymbol{x}^*|D) = \int p(\boldsymbol{x}^*|\boldsymbol{\theta}) p(\boldsymbol{\theta}|D) d\boldsymbol{\theta}$, $p(\boldsymbol{x}^*|\boldsymbol{\theta})$: likelihood (weighted ensemble method, often uses samples of $p(\boldsymbol{\theta}|D)$)
- Prior should express belief and domain knowledge $\xrightarrow{\text{ML}}$ weights should be small $\rightarrow p(\boldsymbol{\theta}) = N(\boldsymbol{\theta}|\boldsymbol{0}, \lambda^{-1}\boldsymbol{I})$ , $\lambda$: precision of the prior
- Completing the square: Bring exponent in canonical squared form: $\exp(-\frac{1}{2}a\mu^2 + b\mu + \text{const}) \rightarrow$ for gaussian distributions: $\mu_N = a^{-1}b$, $\sigma_N^2 = a^{-1}$
- Posterior if prior/likelihood are gaussian: $\mu_N = \frac{N\sigma_0^2}{N\sigma_0^2+\sigma^2}\mu_{\text{ML}} + \frac{\sigma^2}{N\sigma_0^2+\sigma^2}\mu_0$ and $\sigma_N^2 = \frac{\sigma^2\sigma_0^2}{N\sigma_0^2+\sigma^2}$ with $\mu_{\text{ML}} = \frac{\sum x_i}{N}$ and $\mu_0, \sigma_0$ from prior (variance decreases with more training samples, posterior interpolates between prior mean/sample average)
- Gaussian Propagation: predictive distribution is gaussian with $\mu_{x^*} = \mu_N$ and $\sigma_{x^*}^2 = \sigma_N^2 + \sigma^2$
- Conjugate prior for likelihood function $\Leftrightarrow$ posterior/prior: same distribution family
- Bayesian Learning: For large datasets: point estimate, advantage for small dataset
- Simplification: Maximum a-posteriori solution: $\boldsymbol{\theta}_{\text{MAP}} = \text{argmax} \log p(D|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$ maximizes posterior $\rightarrow$ use for prediction: $p(\boldsymbol{x}^*|D) \approx p(\boldsymbol{x}^*|\boldsymbol{\theta}_{\text{MAP}})$
- MAP regression: gaussian prior $\leftrightarrow$ L2; gaussian likelihood $\leftrightarrow$ squared loss $\Leftrightarrow$ ridge regression ($\lambda_{\text{ridge}} = \lambda\sigma^2$, uncertainty only depends on estimated noise $\sigma^2$)
- Bayesian linear regression: likelihood: $p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) = N(\boldsymbol{y}|\boldsymbol{\Phi}\boldsymbol{w}, \sigma^2\boldsymbol{I})$ (multivariate distribution, $\sigma^2$: noise variance), prior: $p(\boldsymbol{w}) = N(\boldsymbol{w}|\boldsymbol{0}, \lambda^{-1}\boldsymbol{I})$
- Gaussian Bayes Rule 1/2 for evaluating $p(\boldsymbol{x}|\boldsymbol{y})$ (different derivations of posterior distribution): rule 1 if $\dim(\boldsymbol{y}) < \dim(\boldsymbol{x})$ (infinite dimensional features), otherwise rule 2

- Gaussian Propagation to evaluate $p(\boldsymbol{y})$
- Posterior/predictive mean equivalent to MAP estimate, but we get uncertainty for parameters: $\boldsymbol{\Sigma}_{w|X,y} = \sigma_y^2(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \sigma_y^2\lambda\boldsymbol{I})^{-1}$ and variance $\sigma^2(\boldsymbol{x}^*)$ is input dependent
- Compute predictive distribution with gaussian propagation
- Gaussian Process: distribution over functions $f(\boldsymbol{x})$ so that any set $\boldsymbol{t}$ of function values evaluated at $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ is jointly gaussian distributed: $f(\boldsymbol{x}) \sim GP(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'}))$, $m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})]$: mean (prior belief about function, zero for simplicity), $k(\boldsymbol{x}, \boldsymbol{x'}) = \mathbb{E}[f(\boldsymbol{x})f(\boldsymbol{x'})]$: positive definite correlation of function evaluations at $\boldsymbol{x}, \boldsymbol{x'}$
- $p(\boldsymbol{t}|\boldsymbol{X}) = N(\boldsymbol{t}|\boldsymbol{0}, \boldsymbol{K}) \xrightarrow{\text{noise}} p(\boldsymbol{y}|\boldsymbol{X}) = N(\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{K} + \sigma_y^2\boldsymbol{I})$ $(y_i = f(\boldsymbol{x}_i) + \varepsilon)$
- Predictive: $\mu(\boldsymbol{x}^*) = \boldsymbol{k}(\boldsymbol{x}^*)^\top(\boldsymbol{K} + \sigma_y^2\boldsymbol{I})^{-1}\boldsymbol{y}$, $\sigma(\boldsymbol{x}^*) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) + \sigma_y - \boldsymbol{k}(\boldsymbol{x}^*)^\top(\boldsymbol{K} + \sigma_y^2\boldsymbol{I})^{-1}\boldsymbol{k}(\boldsymbol{x}^*)$ (mean $\leftrightarrow$ kernel ridge regression + input dependent variance estimate (small for high kernel activations)) $\rightarrow$ kernel-version of bayesian linear regression
- Weight space view: Bayesian: subsume prior precision $\lambda$ into kernel (vector/matrix): $\boldsymbol{K} = \lambda^{-1}\boldsymbol{\Phi}_X\boldsymbol{\Phi}_X^\top$
- Function view (from Gaussian process) vs. weight space view (from Bayesian Linear Regression with kernel trick)
- Posterior derived from Bayesian view (rule 1): $\boldsymbol{\mu}_{w|X,y}, \boldsymbol{\Sigma}_{w|X,y}$ (potentially infinite dimensions) $\rightarrow$ can evaluate predictive distribution with kernel trick (same as $GP$)
- GP: computationally hard ($O(N^3)$), very principled approach to regression learning
- Kernel parameters: weight precision $\lambda$, observation noise $\sigma_y$, length scale $l$ (can be different per dimension): Gaussian Kernel: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \lambda^{-1}\exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||}{2l^2}) + \delta_{ij}\sigma_y^2$
- Optimization: non-convex log-likelihood of data $\rightarrow$ gradient descent (overfitting)
- GP: non-parametric Bayesian approach, prediction equations in closed form (gaussian), hyperparameter optimization complex, outperforms NNs for small datasets

# 10 Neural Networks

- Artificial neuron: $y = \phi(\boldsymbol{w}^\top\boldsymbol{x} + b)$ (like logistic regression)
- Feedforward network: directed acyclic graph (units grouped into layers)
- Fully connected layer ($N$ inputs to $M$ outputs): $\boldsymbol{y} = \phi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$, $\boldsymbol{W} \in \mathbb{R}^{M\times N}$
- Activations: $\sigma$ (0 to 1, kills gradient, not zero-centered (important for initialization), exp computationally hard), tanh (-1 to 1, zero centered, kills gradient), ReLU (fast computation/convergence, not zero centered, $x < 0$: no gradient), leaky ReLU (fast), ELU ($\alpha(e^x - 1)$ for $x < 0$, benefits of ReLU, closer to zero mean, exp hard)

- Each layer computes function: $\boldsymbol{y} = f^L \circ \cdots \circ f^1(\boldsymbol{x})$ (composite of functions)
- XOR: classic example why multiple layers are needed
- Linear layers $\Leftrightarrow$ one linear layer $\rightarrow$ need non-linearities $\rightarrow$ FF-NNs can approximate any function (theoretically with single layer, but exponential number of units)
- Deterministic regression: $\boldsymbol{f} = \boldsymbol{W}^{(L)}\boldsymbol{h}^{(L-1)} + \boldsymbol{b}^{(L)},\ l_i(\boldsymbol{x}_i, \boldsymbol{\theta}) =$ squared loss / Probabilistic: $p(\boldsymbol{y}|\boldsymbol{x}) = N(\boldsymbol{y}|\boldsymbol{W}^{(L)}\boldsymbol{h}^{(L-1)} + \boldsymbol{b}^{(L)}, \boldsymbol{\Sigma}),\ l_i(\boldsymbol{x}_i, \boldsymbol{\theta}) = -\log N(\boldsymbol{y}_i|\boldsymbol{\mu}(\boldsymbol{x}_i), \boldsymbol{\Sigma}(\boldsymbol{x}_i))$
- Deterministic classification: $f = \boldsymbol{W}^{(L)}\boldsymbol{h}^{(L-1)} + b^{(L)},\ l_i(\boldsymbol{x}_i, \boldsymbol{\theta}) =$ hinge loss / Probabilistic: $f = \sigma(\boldsymbol{W}^{(L)}\boldsymbol{h}^{(L-1)} + b^{(L)}),\ l_i(\boldsymbol{x}_i, \boldsymbol{\theta}) = -c_i \log f(\boldsymbol{x}_i) - (1 - c_i)\log(1 - f(\boldsymbol{x}_i))$
- Deterministic multi-class: $\boldsymbol{f} = \boldsymbol{W}^{(L)}\boldsymbol{h}^{(L-1)} + \boldsymbol{b}^{(L)}$, loss not covered / Probabilistic: $\boldsymbol{f} = \text{softmax}(\boldsymbol{W}^{(L)}\boldsymbol{h}^{(L-1)} + \boldsymbol{b}^{(L)}),\ l_i(\boldsymbol{x}_i, \boldsymbol{\theta}) = -\sum_k \boldsymbol{h}_{c_i,k} \log f_k(\boldsymbol{x}_i)$
- NNs learn features that can be seperated linearly by last layer
- Back-propagation learning algorithm: Compute $\frac{\partial L}{\partial \boldsymbol{W}^{(l)}}, \frac{\partial L}{\partial \boldsymbol{b}^{(l)}}$ recursively (chain rule)
- Computation graph: node: input, edge: node computed as function of other node
- Forward pass: compute loss / backward pass: compute derivatives
- Notation: $\overline{y} = \frac{\partial L}{\partial y}$ (error signals)
- Multivariate chain rule: $\frac{\partial}{\partial t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t} = \overline{x}\frac{\partial x}{\partial t} + \overline{y}\frac{\partial y}{\partial t} = \overline{t} \rightarrow$ in vector notation: $\frac{\partial}{\partial t} f(\boldsymbol{x}(t)) = \frac{\partial f}{\partial \boldsymbol{x}}^\top \frac{\partial \boldsymbol{x}}{\partial t}$
- Backpropagation ($v_1, \ldots, v_N$ in topological order): $\forall i$: Compute $v_i$ as a function of $\text{Pa}(v_i)$ (forward pass) $\rightarrow \overline{v_N} = 1 \rightarrow \forall i\ :\ \overline{v_i} = \sum_{j \in \text{Ch}(v_i)} \overline{v_j}\frac{\partial v_j}{\partial v_i}$ (backward pass)
- Chain rule for matrix-vector products: $\nabla_W f = \frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{W}} = \frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\boldsymbol{x}^\top\ (\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$
- Forward: one add-multiply operation per weight / backward: two times forward (cost linear in number layers, quadratic in units per layer)
- Backprop neurally implausible (biological alternatives much slower on computers)
- Problems with standard SGD: slow along shallow dimension, jitter along steep dimension; stuck in local minima; noisy loss function due to mini-batches
- Momentum term (running gradient average): $\boldsymbol{m}_{k+1} = \gamma_k \boldsymbol{m}_k + (1 - \gamma_k)\nabla L$ / Geometric Average: $\boldsymbol{m}_k = (1 - \gamma)\sum_i \gamma^{k-i}\boldsymbol{g}_i$
- Gradient normalization (RMSProp): large steps in plateaus, small steps in steep areas: $\boldsymbol{g}_k = \nabla_\theta L(\boldsymbol{\theta}_k),\ \boldsymbol{b}_{k+1,i} = \gamma \boldsymbol{v}_{k,i} + (1 - \gamma)\boldsymbol{g}_{k,i}^2, \boldsymbol{\theta}_{k+1,i} = \boldsymbol{\theta}_{k,i} - \frac{\eta}{\sqrt{\boldsymbol{v}_{k+1,i} + \varepsilon}}\boldsymbol{g}_{k,i}$ per dimension $i$, $\boldsymbol{v}_k$: average of gradient norms, $\varepsilon$: prevent division by zero

- Adam: adaptive momentum + normalization: $\boldsymbol{\theta}_{k+1,i} = \boldsymbol{\theta}_{k,i} - \frac{\eta c_2(k)}{\sqrt{c_1(k)\boldsymbol{v}_{k+1,i}+\varepsilon}}\boldsymbol{m}_{k+1,i}$ (no convergence guarantee, underestimation at start fixed by $c_i(k) = \frac{1}{1-\gamma_i^k}$
- Learning rate decay: Reduce at fixed points / Cosine: $\alpha_t = \frac{1}{2}\alpha_0(1 + \cos(\frac{t\pi}{T}))$ / Linear $\alpha_t = \alpha_0(1 - \frac{t}{T})$ / Inverse root: $\alpha_t = \frac{\alpha_0}{\sqrt{t}}$ ($T$: total number of epochs)
- First order optimization: step in direction of minimum of linear approximation / second order optimization: step to minimum of quadratic approximation
- $\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \frac{1}{2}\boldsymbol{H}^{-1}\boldsymbol{g}$ with Hessian $H = \nabla_\theta^2 L(\boldsymbol{\theta})$ (no hyperparameters, no learning rate, less iterations, inverse in $O(N^3)$, $N$ is in the millions)
- Solutions: quasi-Newton methods (BFGS, approximate Hessian over time) / Limited memory BFGS (does not store full $\boldsymbol{H}^{-1}$, works well in full batch) → in practice use Adam or L-BFGS (only on full batch with small noise)
- Regularization: model ensembles: train multiple models (or use snapshots of one during training), average their results
- Dropout: randomly (often 50%) set neurons to zero (in each forward pass) → forces redundancy, can be interpreted as ensembles with shared parameters
- Testing dropout: average over multiple dropout masks (ensemble view) / multiply each weight by dropout rate (expectation view)
- Drop connect: drop neuron connections (training) / use all connections (testing)
- Data preprocessing: initialization optimized for zero-mean unit variance data / PCA / whitening of low-d data (covariance matrix is $\boldsymbol{I}$)
- Classification loss less sensitive to small weight changes after normalization
- Weight initialization: constant → all gradients equal, no distinct features can be learned → random initialization needed
- Fixed variance → activations go to zero/saturate over deep layers (no gradients)
- Xavier intialization ($\sigma_W = \frac{1}{\sqrt{D_{\text{in}}}}$): activations nicely scaled for all layers (for tanh) / For ReLU: $\sigma_W = \frac{2}{\sqrt{D_{\text{in}}}}$
- Practice tips
  1. Check initial loss (without L2 should be $\log C$ for softmax with $C$ classes)
  2. Overfit small sample (get 100% training accuracy, change architecture / $\eta$)
  3. Find $\eta$ to strongly decrease loss in 100 iterations (full training data, small L2)
  4. Grid search around $\eta$ / L2 from previous step (train each for 1 to 5 epochs)
  5. Train best models from step 4 for longer (10 to 20 epochs) without $\eta$ decay

6. Loss curves: plateau end $\rightarrow \eta$ decay / plateau beginning $\rightarrow$ bad initialization / plateau after $\eta$ step decay $\rightarrow$ decay later / validation accuracy going up $\rightarrow$ train longer / validation accuracy going down $\rightarrow$ regularize/more data / same training/validation accuracy $\rightarrow$ train longer/bigger model

- NNs work very well, even though we have more parameters than training samples

## 11 Convolutional and Recurrent Neural Networks

- Image inputs $\rightarrow$ huge amount of weights with FC-layers
- Close pixels more correlated $\rightarrow$ use convolutions (slide filter over image)
- Stack filters to obtain multi-channel output
- Stride $S$: step-size ($> 1 \rightarrow$ down-sampling) / (zero)-padding $P$: fill image borders
- Convolutional Layer: $W_1 \times H_1 \times D_1 \rightarrow W_2 \times H_2 \times D_2$ with $W_2 = (W_1 - F + 2P)/S + 1$, $H_2 = (H_1 - F + 2P)/S + 1$ and $D_2 = K$: number of filters, $F$: kernel size
- (Max)-Pooling: smaller output dimension (applied to each channel with $P = 0$)
- Convolutional network: Convolutional layers, activations, pooling, FC layers at end
- Optimize deep models: residual block computes $F(x) + x \rightarrow$ new layers do no harm with $F(x) = 0$ at beginning
- Transfer learning (for small datasets): Convolutional layers are generic $\rightarrow$ reuseable (only train last FC layer(s) and freeze rest)
- AlexNet (2012): first use of ReLU, 8 layers (first CNN winner of ImageNet) / VGG (2014): more smaller filters (more non-linearities, fewer parameters, 19 layers) / ResNet (2015): very deep using residual connections (152 layers)
- Recurrent NNs: one to many (image captioning) / many to one (sentiment) / many to many (translation, video classification) $\rightarrow$ use old state as input
- State $\boldsymbol{h}_t = f_W(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) = \tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t)$, $\boldsymbol{y}_t = \boldsymbol{W}_{hy}\boldsymbol{h}_t$
- Computational graph: unroll time steps $\rightarrow$ network depth $T$, reuse $\boldsymbol{W}$ each step
- Backpropagation through time (BPTT): forward/backward through entire sequence
- Truncated BPTT: keep $\boldsymbol{h}$, but only backpropagate for smaller number of steps
- Image Captioning: $\boldsymbol{h}_t = \tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{ih}\boldsymbol{v})$ with $\boldsymbol{x}$ : previous word, $\boldsymbol{v}$ : last layer CNN
- $\nabla \boldsymbol{h}$ depends on $\boldsymbol{W}$: largest singular value $> 1$: exploding gradients $\rightarrow$ scale gradient / largest singular value $< 1$: vanishing gradients $\rightarrow$ different RNN architecture

- Long-term short-term memory (LSTM): gated contribution of state/input (forget (erase cell), input (write), $g$ (how much to write), output (how much to reveal))
- LSTM: Backpropagation from $c_t$ to $c_{t-1}$: only elementwise multiplication (uninterrupted gradient flow similar to ResNet)
- Stack: layer 1 output sequence $\rightarrow$ layer 2 input (only dropout non-recurrent edges)
- Gated Recurrent Units: no explicit $f$ gate, less parameters, similar performance

# 12 Wrap-Up

| Chapter | Classical Supervised Learning | Classical Unsupervised Learning | Kernel Methods | Bayesian Learning | Neural Networks |
|---|---|---|---|---|---|
| Algorithms | *Regression*: Linear, Ridge, KNN, Trees, Forests *Classification*: Logistic Regression, KNN, Trees, Forests | (p)PCA Clustering: Agglomerative, K-Means, EM for GMMs Density Estimation: KDE, KNN, Mixture Models | Kernel Regression, SVMs | Bayesian Linear Regression, Gaussian Processes | FF-NNs, Backprop, CNNs, LSTMs |
| Basics | Matrix and Vector Calculus, Probability Theory, MLE, Gradient Descent | Constraint Optimization, EM | Sub-gradients, Constraint Optimization | "Completing the square", Gaussian Conditioning | Most of the others |
| Representations | Features, Basis Functions, Instances, Trees | Instances, Linear Projections, Centroids, Mixture Models | Kernels | Features, Kernels | NNs |
| Optimization | Least-squares, Gradient Descent | Eigen-Value Decomposition, EM | Sub-gradients, Quadratic Solver | Computing the Posterior | Adam, 2nd Order Gradient, Sub-gradients (ReLU) |
| Loss | MSE/SSE, Gaussian Log-Likelihood, Binary Cross Entropy, Soft-Max Likelihood | Reproduction Error, SSD, Sum of Discrepancies, Marginal Log-Likelihood | Maximum Margin, Hinge Loss | MAP, Posterior Approximation | Most of the others |