Regression

Terminology

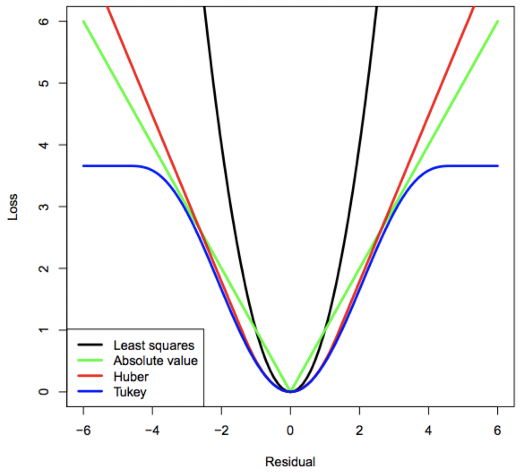
- Data consists of **pairs** $(\mathbf{x}_n, y_n)$, where y_n is the n'th output and x_n is a vector of D inputs. The number of pairs N is the data-size and D is the dimensionality.
- Two goals of regression: **prediction** and **interpretation**
- The regression function: $y_n \approx f_w(\mathbf{x}_n)\ \forall n$
- Regression finds correlation not a causal relationship.
- **Input variables** a.k.a. covariates, independent variables, explanatory variables, exogenous variables, predictors, regressors.
- **Output variables** a.k.a. target, label, response, outcome, dependent variable, endogenous variables, measured variable, regressands.

Linear Regression

- Assumes linear relationship between inputs and output.
- $y_n \approx f(\mathbf{x}_n) := w_0 + w_1 x_{n1} + ... + w_D x_{nD} := \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}$ contain the additional offset term (a.k.a. bias).
- Given data we learn the weights $\mathbf{w}$ (a.k.a. estimate or fit the model)
- Overparameterisation $D > N$ eg. univariate linear regression with a single data point $y_1 \approx w_0 + w_1 x_{11}$. This makes the task under-determined (no unique solution).

Loss Functions $\mathcal{L}$

- A loss function (a.k.a. energy, cost, training objective) quantifies how well the model does (how costly its mistakes are).
- $y \in \mathbb{R} \Rightarrow$ desirable for cost to be symmetric around 0 since $\pm$ errors should be penalized equally.
- Cost function should penalize "large" mistakes and "very large" mistakes similarly to be robust to outliers.
- Mean Squared Error:
$\mathsf{MSE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^N [y_n - f_{\mathrm{w}}(\mathbf{x}_n)]^2$
not robust to outliers.
- Mean Absolute Error:
$\mathsf{MAE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^N |y_n - f_{\mathrm{w}}(\mathbf{x}_n)|$
- Convexity: a function is convex iff a line segment between two points on the function's graph always lies above the function.
- Convexity: a function $h(\mathbf{u}), \mathbf{u} \in \mathbb{R}^D$ is convex if $\forall\ \mathbf{u}, \mathbf{v} \in \mathbb{R}^D, 0 \le \lambda \le 1$:
$h(\lambda\mathbf{u} + (1-\lambda)\mathbf{v}) \le \lambda h(\mathbf{u}) + (1-\lambda) h(\mathbf{v})$
Strictly convex if $\le \Rightarrow <$
- Convexity, a desired computational property: A strictly convex function has a unique global minimum $\mathbf{w}^*$. For convex functions, every local minimum is a global minimum.
- Sums of convex functions are also convex $\Rightarrow$ MSE combined with a linear model is convex in $\mathbf{w}$.
- Proof of convexity for MAE:
$\mathsf{MAE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^N \mathcal{L}_n(\mathbf{w}), \mathcal{L}_n(\mathbf{w}) = |y_n - f_{\mathrm{w}}(\mathbf{x}_n)|$
$\mathcal{L}_n(\lambda w_1 + (1-\lambda)w_2) \le \lambda\mathcal{L}_n(w_1) + (1-\lambda)\mathcal{L}_n(w_2)$
$|y_n - x_n^T(\lambda w_1 + (1-\lambda)w_2)| \le \lambda|y_n - x_n^T w_1| + (1-\lambda)|y_n - x_n^T w_2|$
$(1-\lambda) \ge 0 \Rightarrow (1-\lambda)|y_n - x_n^T w_2| = |(1-\lambda)y_n - (1-\lambda)x_n^T w_2|$
$a = \lambda y_n - \lambda x_n^T w_1, b = (1-\lambda)y_n - (1-\lambda)x_n^T w_2$
$a + b = y_n - x_n^T(\lambda w_1 + (1-\lambda)w_2)$
$|a+b| \le |a| + |b| \Rightarrow \mathcal{L}_n(\mathbf{w})$ convex $\Rightarrow \mathsf{MAE}(\mathbf{w})$ convex
- Huber loss:
$\mathcal{H}uber(e) := \begin{cases} \frac{1}{2}e^2 & ,\text{if } |e| \le \delta \\ \delta|e| - \frac{1}{2}\delta^2 & ,\text{if } |e| > \delta \end{cases}$ convex, differentiable, and robust to outliers but setting δ is not easy.
- Tukey's bisquare loss:
$\frac{\partial\mathcal{L}}{\partial e} := \begin{cases} e\{1 - e^2/\delta^2\}^2 & ,\text{if } |e| \le \delta \\ 0 & ,\text{if } |e| > \delta \end{cases}$ non-convex, but robust to outliers.



Optimisation

- Given $\mathcal{L}(\mathbf{w})$ we want $\mathbf{w}^* \in \mathbb{R}^D$ which minimises the cost: $\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \rightarrow$ formulated as an optimisation problem
- Local minimum $\mathbf{w}^* \Rightarrow \exists\epsilon > 0$ s.t.
$\mathcal{L}(\mathbf{w}^*) \le \mathcal{L}(\mathbf{w})\ \forall\mathbf{w}$ with $\|\mathbf{w} - \mathbf{w}^*\| < \epsilon$
- Global minimum $\mathbf{w}^*$, $\mathcal{L}(\mathbf{w}^*) \le \mathcal{L}(\mathbf{w})\ \forall\mathbf{w} \in \mathbb{R}^D$

Smooth Optimization

- A gradient is the slope of the tangent to the function. It points to the direction of largest increase of the function.
$\nabla\mathcal{L}(\mathbf{w}) := \left[\frac{\partial\mathcal{L}(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial\mathcal{L}(\mathbf{w})}{\partial w_D}\right]^\top \in \mathbb{R}^D$

Gradient Descent

- To minimize the function, we iteratively take a step in the opposite direction of the gradient

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\nabla\mathcal{L}\left(\mathbf{w}^{(t)}\right)$$

where $\gamma > 0$ is the step-size (or learning rate). Then repeat with the next t.
- Example: Gradient descent for 1parameter model to minimize MSE:
$f_w(x) = w_0 \Rightarrow \mathcal{L}(w) = \frac{1}{2N}\sum_{n=1}^N (y_n - w_0)^2 = \nabla\mathcal{L} = \frac{\partial}{\partial w_0}\mathcal{L} = \frac{1}{2N}\sum -2(y_n - w_0) = (-\frac{1}{N}\sum y_n) + w_0 = w_0 - \bar{y}$
$(min_w L(w) = w_0 - \bar{y} = 0 \Rightarrow w_0 = \bar{y})$
$w_0^{(t+1)} := (1-\gamma)w_0^{(t)} + \gamma\bar{y}$
where $\bar{y} := \sum_n y_n/N$. When is this sequence guaranteed to converge? When $\gamma > 2$ you start having an exploding GD.

Gradient Descent for Linear MSE

We define the error vector $\mathbf{e}$: $\mathrm{e} = \mathbf{y} - \mathbf{X}\mathbf{w}$ and MSE as follows:
$\mathcal{L}(\mathbf{w}) := \frac{1}{2N}\sum_{n=1}^N \left(y_n - \mathbf{x}_n^\top\mathbf{w}\right)^2 = \frac{1}{2N}\mathbf{e}^\top\mathbf{e}$
then the gradient is given by
$\nabla\mathcal{L}(\mathbf{w}) = -\frac{1}{N}\mathbf{X}^\top\mathbf{e}$
Computational cost: $\Theta(N \times D)$

Stochastic Gradient Descent

$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^N \mathcal{L}_n(\mathbf{w})$
$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\nabla\mathcal{L}_n\left(\mathbf{w}^{(t)}\right)$
- Compuational cost: $\Theta(D)$
- Cheap and unbiased estimate of the gradient!
$E[\nabla\mathcal{L}_n(w)] = \frac{1}{n}\sum_{n=1}^N \nabla\mathcal{L}_n(w) = \nabla\left(\frac{1}{N}\sum\cdots\right) = \nabla\mathcal{L}(n)$ which is the true gradient direction.

Mini-batch SGD

$\mathbf{g} := \frac{1}{|B|}\sum_{n\in B}\nabla\mathcal{L}_n\left(\mathbf{w}^{(t)}\right)$
$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\mathbf{g}$.
- Randomly chosen a subset $B \subseteq [N]$ of the training examples. For each of these selected examples n, we compute the respective gradient $\nabla\mathcal{L}_n$, at the same current point $\mathbf{w}^{(t)}$.
- The computation of $\mathbf{g}$ can be parallelized easily. This is how current deep-learning applications utilize GPUs (by running over $|B|$ threads in parallel).
- $B := [N]$, we obtain $\mathbf{g} = \nabla\mathcal{L}$.

Non-Smooth Optimization

- An alternative characterization of convexity, for differentiable functions is given by
$\mathcal{L}(\mathbf{u}) \ge \mathcal{L}(\mathbf{w}) + \nabla\mathcal{L}(\mathbf{w})^\top(\mathbf{u} - \mathbf{w}) \quad \forall\mathbf{u}, \mathbf{w}$
meaning that the function must always lie above its linearization.

Subgradients

- A vector $\mathbf{g} \in \mathbb{R}^D$ such that
$\mathcal{L}(\mathbf{u}) \ge \mathcal{L}(\mathbf{w}) + \mathbf{g}^\top(\mathbf{u} - \mathbf{w}) \quad \forall\mathbf{u}$
is called a subgradient to the function $\mathcal{L}$ at $\mathbf{w}$.
- This definition makes sense for objectives $\mathcal{L}$ which are not necessarily differentiable (and not even necessarily convex).
- If $\mathcal{L}$ is convex and differentiable at $\mathbf{w}$, then the only subgradient at $\mathbf{w}$ is $\mathbf{g} = \nabla\mathcal{L}(\mathbf{w})$.

Subgradient Descent

$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\mathbf{g}$
for $\mathbf{g}$ a subgradient to $\mathcal{L}$ at the current iterate $\mathbf{w}^{(t)}$.

Example: Optimizing Linear MAE

1) Compute a subgradient of the absolute value function $h : \mathbb{R} \rightarrow \mathbb{R}, h(e) := |e|$.
$g = \begin{cases} -1 & \text{if } e < 0 \\ [-1, 1] & \text{if } e = 0 \\ 1 & \text{if } e > 0 \end{cases}$
2) Recall the definition of the mean absolute error:
$\mathcal{L}(\mathbf{w}) = \mathrm{MAE}(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^N |y_n - f_{\mathbf{w}}(\mathbf{x}_n)|$
For linear regression, its (sub)gradient is easy to compute using the chain rule. Compute it!
The subgradient is given by:
$\partial\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^N g(e_n)\nabla f_{\mathbf{w}}(\mathbf{x}_n) = \frac{1}{N}\sum_{n=1}^N g(e_n)\mathbf{x}_n$
since $f_{\mathbf{w}}(\mathbf{x}_n) = \mathbf{w}^T\mathbf{x}_n$, then $\nabla f_{\mathbf{w}}(\mathbf{x}_n) = \mathbf{x}_n$ where $e_n = y_n - f_{\mathbf{w}}(\mathbf{x}_n)$.
See Exercise Sheet 2.

Stochastic Subgradient Descent

- still abbreviated SGD commonly.
- Same, $\mathbf{g}$ being a subgradient to the randomly selected $\mathcal{L}_n$ at the current iterate $\mathbf{w}^{(t)}$.
- SGD update for linear MAE:
$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \gamma g(e_n)\mathbf{x}_n$

Variants of SGD

SGD with Momentum

pick a stochastic gradient $\mathbf{g}$
$\mathbf{m}^{(t+1)} := \beta_1\mathbf{m}^{(t)} + (1-\beta_1)\,\mathbf{g}$ (momentum term)
$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\mathbf{m}^{(t+1)}$
- momentum from previous gradients (acceleration)

Adam

- pick a stochastic gradient $\mathbf{g}$
$\mathbf{m}^{(t+1)} := \beta_1\mathbf{m}^{(t)} + (1-\beta_1)\,\mathbf{g}$
$\mathbf{v}_i^{(t+1)} := \beta_2\mathbf{v}_i^{(t)} + (1-\beta_2)\,(\mathbf{g}_i)^2 \quad \forall i$ (Momentum t
$\mathbf{w}_i^{(t+1)} := \mathbf{w}_i^{(t)} - \frac{\gamma}{\sqrt{\mathbf{v}_i^{(t+1)}}}\mathbf{m}_i^{(t+1)} \quad \forall i$
- faster forgetting of older weights
- is a momentum variant of Adagrad
- coordinate-wise adjusted learning rate

-strong performance in practice, e.g. for self-attention networks

SignSGD

pick a stochastic gradient $\mathbf{g}$

$\mathbf{w}_i^{(t+1)} := \mathbf{w}_i^{(t)} - \gamma \operatorname{sign}(\mathbf{g}_i)$

- only use the sign (one bit) of each gradient entry $\rightarrow$ communication efficient for distributed training
- convergence issues

Constrained Optimization

- Sometimes, optimization problems come posed with additional constraints:

$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}), \quad$ subject to $\mathbf{w} \in \mathcal{C}$.

- The set $\mathcal{C} \subset \mathbb{R}^D$ is called the constraint set.

Convex Sets

A set $\mathcal{C}$ is convex iff the line segment between any two points of $\mathcal{C}$ lies in $\mathcal{C}$, i.e., if for any $\mathbf{u}, \mathbf{v} \in \mathcal{C}$ and any θ with $0 \leq \theta \leq 1$, we have

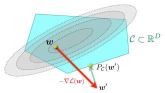
$\theta \mathbf{u} + (1-\theta)\mathbf{v} \in \mathcal{C}$

- Intersubsections of convex sets are convex - Projections onto convex sets are unique.(and often efficient to compute)
- Formal definition:

$P_{\mathcal{C}}(\mathbf{w}') := \arg\min_{\mathbf{v} \in \mathcal{C}} \|\mathbf{v} - \mathbf{w}'\|$.

Projected Gradient Descent

- Idea: add a projection onto $\mathcal{C}$ after every step:



$P_{\mathcal{C}}(\mathbf{w}') := \arg\min_{\mathbf{v} \in \mathcal{C}} \|\mathbf{v} - \mathbf{w}'\|$.

$\mathbf{w}^{(t+1)} := P_{\mathcal{C}}\left[\mathbf{w}^{(t)} - \gamma \nabla \mathcal{L}\left(\mathbf{w}^{(t)}\right)\right]$

- Projected SGD. Same SGD step, followed by the projection step. Same convergence properties.

Constrained $\rightarrow$ Unconstrained Problems

$\min_{w \in C} \mathcal{L}(w) \sim \min_w \mathcal{L}(w) + P(w)$

- Alternatives to projected gradient methods
- Use penalty functions instead of directly solving $\min_{\mathbf{w} \in \mathcal{C}} \mathcal{L}(\mathbf{w})$.
- "brick wall" (indicator function)

$$P(w) = I_{\mathcal{C}}(\mathbf{w}) := \begin{cases} 0 & \mathbf{w} \in \mathcal{C} \\ \infty & \mathbf{w} \notin \mathcal{C} \end{cases}$$

Can't run GD because non-continuous objective.

- Penalize error. Example:

$\mathcal{C} = \{\mathbf{w} \in \mathbb{R}^D \mid A\mathbf{w} = \mathbf{b}\} \Rightarrow P(w) = \lambda \|A\mathbf{w} - \mathbf{b}\|^2$

- ℓ_1 norm encourages sparsity which reduces memory: $P(w) = \lambda \|\mathbf{w}\|_1$
- Linearized Penalty Functions (see Lagrange Multipliers)

Implementation Issues

1) Stopping criteria: $\nabla \mathcal{L}(\mathbf{w}) \approx 0$

2) Optimality:

<u>1st order optimiality condition:</u> if $\mathcal{L}$ is convex and $\nabla \mathcal{L}(w^*) = 0 \Rightarrow$ global optimality

<u>2nd order</u>: $\mathcal{L}$ potentially non-convex, if $\nabla \mathcal{L}(w^*) = 0$ & $\nabla^2 \mathcal{L}(w^*) \geq 0 \Rightarrow w$ local minimum.

$\nabla^2 \mathcal{L}(\mathbf{w}) := \frac{\partial^2 \mathcal{L}}{\partial \mathbf{w} \partial \mathbf{w}^\top}(\mathbf{w})$ is expensive.

3) Step-size selection: $\gamma >>$ might diverge. $\gamma <<$ slow convergence. Convergence to local minimum guaranteed only when $\gamma < \gamma_{\min}$, $\gamma_{\min}$ depends on the problem.

4) Line-search methods: For some $\mathcal{L}$, set step-size automatically.

5) Feature normalization: GD is sensitive to ill-conditioning $\rightarrow$ Normalize your input features i.e. pre-condition the optimization problem.

Non-Convex Optimization

- Real-world problems are not convex!
- All we have learnt on algorithm design and performance of convex algorithms still helps us in the nonconvex world.

SGD Theory

- For convergence, $\gamma^{(t)} \rightarrow 0$ "appropriately". Robbins-Monroe condition suggests to take $\gamma^{(t)}$ s.t.:

$\sum_{t=1}^{\infty} \gamma^{(t)} = \infty, \quad \sum_{t=1}^{\infty} \left(\gamma^{(t)}\right)^2 < \infty$

Example: $\gamma^{(t)} := 1/(t+1)^r$ where $r \in (0.5, 1)$.

Least Squares

- Linear regression + MSE $\rightarrow$ compute the optimum of the cost function analytically by solving a linear system of D equations (normal equations)
- Derive the normal equations, proove convexity, optimality conditions for convex functions ($\nabla \mathcal{L}(\mathbf{w}^\star) = \mathbf{0}$.)

Normal Equations

$\mathcal{L}(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2$

$= \frac{1}{2N}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$,

- Proof of convexity:

1) Simplest way: observe that $\mathcal{L}$ is naturally represented as the sum (with positive coefficients) of the simple terms $\left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2$. Further, each of these simple terms is the composition of a linear function with a convex function (the square function). Therefore, each of these simple terms is convex and hence the sum is convex.

2) Directly verify the definition, that for any $\lambda \in [0,1]$ and $\mathbf{w}, \mathbf{w}'$

$\mathcal{L}(\lambda \mathbf{w} + (1-\lambda)\mathbf{w}') - (\lambda \mathcal{L}(\mathbf{w}) + (1-\lambda)\mathcal{L}(\mathbf{w}')) \leq 0.$

LHS $= -\frac{1}{2N}\lambda(1-\lambda)\|\mathbf{X}(\mathbf{w} - \mathbf{w}')\|_2^2 < 0$,

3) We can compute the second derivative (the Hessian) and show that it is positive semidefinite (all its eigenvalues are non-negative).

$$\begin{aligned}
\mathbf{H}(\mathbf{w}) &= \frac{1}{2N} \nabla^2 \left((\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w})\right) \\
&= \frac{1}{2N} \nabla \left(-2(\mathbf{y} - \mathbf{X}\mathbf{w})\mathbf{X}^\top\right) \\
&= \frac{-2}{2N} \nabla (\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w})) \\
&= \frac{-1}{N} \mathbf{X}^\top \nabla (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
&= \frac{-1}{N} \mathbf{X}^\top (\nabla \mathbf{y} - \nabla(\mathbf{X}\mathbf{w})) \\
&= \frac{-1}{N} \mathbf{X}^\top (\mathbf{0} - \mathbf{X}) \\
&= \frac{1}{N} \mathbf{X}^\top \mathbf{X}
\end{aligned}$$

Singular value decomposition (SVD): $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$ U and V are orthogonal matrices, and S is a diagonal matrix with the singular values σ_i on the diagonal.

$\mathbf{H}(\mathbf{w}) = \frac{1}{N}\mathbf{X}^\top \mathbf{X} = \frac{1}{N}\mathbf{V}\mathbf{S}^2\mathbf{V}^\top$

where $\mathbf{S}^2$ is a diagonal matrix with the squares of the singular values.

Let $\mathbf{v}$ be a non-zero vector,

$$\begin{aligned}
\mathbf{v}^\top \mathbf{H}(\mathbf{w}) \mathbf{v} &= \frac{1}{N} \mathbf{v}^\top \mathbf{V}\mathbf{S}^2\mathbf{V}^\top \mathbf{v} \\
&= \frac{1}{N} (\mathbf{V}^\top \mathbf{v})^\top \mathbf{S}^2 (\mathbf{V}^\top \mathbf{v}) \\
&= \frac{1}{N} \|\mathbf{S}(\mathbf{V}^\top \mathbf{v})\|^2 \geq 0
\end{aligned}$$

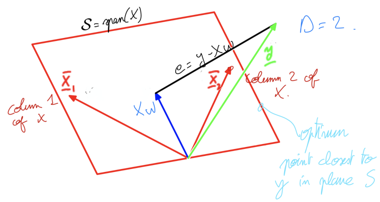
$\mathbf{S}$ diagonal matrix with non-negative entries, $\mathbf{V}^\top \mathbf{v}$ vector.

- Now find its minimum

$\nabla \mathcal{L}(\mathbf{w}) = -\frac{1}{N}\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) = \mathbf{0}$

$\Rightarrow \mathbf{X}^\top \underbrace{(\mathbf{y} - \mathbf{X}\mathbf{w})}_{\text{error}} = \mathbf{0}$

Geometric Interpretation



Closed form

- $\mathbf{X}^\top \mathbf{X} \in \mathbb{R}^{D \times D}$ is called the Gram matrix.
- If invertible, we can get a closed-form expression for the minimum:

$\mathbf{w}^\star = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$

Invertibility and Uniqueness

- $\mathbf{X}^\top \mathbf{X} \in \mathbb{R}^{D \times D}$ invertible iff $\operatorname{rank}(\mathbf{X}) = D$.
- Proof: assume $\operatorname{rank}(\mathbf{X}) < D$. Then there exists a non-zero vector $\mathbf{u}$ so that $\mathbf{X}\mathbf{u} = \mathbf{0}$. It follows that $\mathbf{X}^\top \mathbf{X}\mathbf{u} = \mathbf{0}$, and so $\operatorname{rank}\left(\mathbf{X}^\top \mathbf{X}\right) < D$. Therefore, $\mathbf{X}^\top \mathbf{X}$ is not invertible.

Conversely, assume that $\mathbf{X}^\top \mathbf{X}$ is not invertible. Hence, there exists a non-zero vector $\mathbf{v}$ so that $\mathbf{X}^\top \mathbf{X}\mathbf{v} = \mathbf{0}$. It follows that

$$\mathbf{0} = \mathbf{v}^\top \mathbf{X}^\top \mathbf{X}\mathbf{v} = (\mathbf{X}\mathbf{v})^\top (\mathbf{X}\mathbf{v}) = \|\mathbf{X}\mathbf{v}\|^2$$

This implies that $\mathbf{X}\mathbf{v} = \mathbf{0}$, i.e., $\operatorname{rank}(\mathbf{X}) < D$.

Rank Deficiency and Ill-Conditioning

- In practice, $\mathbf{X}$ is often rank deficient.
- If $D > N$, we always have $\operatorname{rank}(\mathbf{X}) < D$ (since row rank = col. rank)
- If $D \leq N$, but some of the columns $\mathbf{x}_{:}$ are (nearly) collinear, then the matrix is illconditioned, leading to numerical issues when solving the linear system.
- Using a linear system solver, one can solve this problem.

Closed-form solution for MAE

Can you derive closed-form solution for 1-parameter model when using MAE cost function?

Maximum Likelihood

Gaussian distribution and independence

- A Gaussian random variable in $\mathbb{R}$ with mean μ and variance σ^2 has a density of

$p(y \mid \mu, \sigma^2) = \mathcal{N}(y \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right]$.

- The density of a Gaussian random vector with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ (which must be a positive semi-definite matrix) is

$\mathcal{N}(\mathbf{y} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$

$= \frac{1}{\sqrt{(2\pi)^D \det(\boldsymbol{\Sigma})}} \exp\left[-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu})\right]$.

- Two random variables X and Y are called independent when $p(x,y) = p(x)p(y)$.

A probabilistic model for least-squares

- Data generation: $y_n = \mathbf{x}_n^\top \mathbf{w} + \epsilon_n$ where the ϵ_n (the noise) is a zero mean Gaussian random variable.
- Given N samples, the likelihood of the data vector $\mathbf{y} = (y_1, \cdots, y_N)$ given the input $\mathbf{X}$ and the model $\mathbf{w}$ is equal to

$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} p(y_n \mid \mathbf{x}_n, \mathbf{w}) = \prod_{n=1}^{N} \mathcal{N}(y_n \mid \mathbf{x}_n^\top \mathbf{w}, \sigma^2)$.

- The probabilistic view point is that we should maximize this likelihood over the choice of model w. I.e., the "best" model is the one that maximizes this likelihood.

Defining cost with log-likelihood

$\mathcal{L}_{LL}(\mathbf{w}) := \log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = -\frac{1}{2\sigma^2} \sum_{n=1}^{N} \left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2 + \text{cnst}.$

Maximum-likelihood estimator (MLE)

$\arg\min_{\mathbf{w}} \mathcal{L}_{\mathrm{MSE}}(\mathbf{w}) = \arg\max_{\mathbf{w}} \mathcal{L}_{\mathrm{LL}}(\mathbf{w})$.
- MLE $\rightarrow$ finding the model under which the observed data is most likely to have been generated from (probabilistically).

Properties of MLE

- MLE is a sample approximation to the expected log-likelihood:

$$\mathcal{L}_{\mathrm{LL}}(\mathbf{w}) \approx \mathbb{E}_{p(y,\mathbf{x})}[\log p(y \mid \mathbf{x}, \mathbf{w})]$$

- MLE is consistent, i.e., it will give us the correct model assuming that we have a sufficient amount of data. (can be proven under some weak conditions)
$\mathbf{w}_{\mathrm{MLE}} \longrightarrow^{p} \mathbf{w}_{\text{true}}$ in probability
- The MLE is asymptotically normal, i.e.,
$(\mathbf{w}_{\mathrm{MLE}} - \mathbf{w}_{\text{true}}) \longrightarrow^{d} \frac{1}{\sqrt{N}} \mathcal{N}\left(\mathbf{w}_{\mathrm{MLE}} \mid \mathbf{0}, \mathbf{F}^{-1}(\mathbf{w}_{\text{true}})\right)$
where $\mathbf{F}(\mathbf{w}) = -\mathbb{E}_{p(\mathbf{y})}\left[\frac{\partial^2 \mathcal{L}}{\partial \mathbf{w} \partial \mathbf{w}^\top}\right]$ is the Fisher information.
- MLE is efficient, i.e. it achieves the Cramer-Rao lower bound. Covariance $(\mathbf{w}_{\mathrm{MLE}}) = \mathbf{F}^{-1}(\mathbf{w}_{\text{true}})$

Laplace distribution

$p(y_n \mid \mathbf{x}_n, \mathbf{w}) = \frac{1}{2b} e^{-\frac{1}{b}\left|y_n - \mathbf{x}_n^\top \mathbf{w}\right|}$

Regularization

$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) + \Omega(\mathbf{w})$

L_2-Regularization: Ridge Regression

- Standard Euclidean norm (L_2- norm):
$\Omega(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2$ where $\|\mathbf{w}\|_2^2 = \sum_i w_i^2$.
- When $\mathcal{L}$ is MSE, this is called ridge regression:
$\min_{\mathbf{w}} \frac{1}{2N} \sum_{n=1}^N \left[y_n - \mathbf{x}_n^\top \mathbf{w}\right]^2 + \lambda \|\mathbf{w}\|_2^2$
- Least squares is a special case of this: set $\lambda := 0$.

Explicit solution for w:

$\nabla = \nabla \mathcal{L} + \nabla \Omega = -\frac{1}{N} X^T(y - Xw) + \lambda 2w = 0 \Rightarrow$
$\mathbf{w}_{\text{ridge}}^\star = \left(\mathbf{X}^\top \mathbf{X} + \lambda' \mathbf{I}\right)^{-1} \mathbf{X}^\top \mathbf{y}$
(here for simpler notation $\frac{\lambda'}{2N} = \lambda$)

Ridge Regression Fights Ill-Conditioning

- Lifting the eigenvalues : The eigenvalues of $\left(\mathbf{X}^\top \mathbf{X} + \lambda' \mathbf{I}\right)$ are all at least λ' and so the inverse always exists.
- Proof: Write the Eigenvalue decomposition of $\mathbf{X}^\top \mathbf{X}$ as $\mathbf{U} \mathbf{S} \mathbf{U}^\top$. We then have
$\mathbf{X}^\top \mathbf{X} + \lambda' \mathbf{I} = \mathbf{U} \mathbf{S} \mathbf{U}^\top + \lambda' \mathbf{U} \mathbf{I} \mathbf{U}^\top = \mathbf{U}\left[\mathbf{S} + \lambda' \mathbf{I}\right] \mathbf{U}^\top$
- Alternative proof: Recall that for a symmetric matrix $\mathbf{A}$ we can also compute eigenvalues by looking at the so-called Rayleigh ratio,
$R(\mathbf{A}, \mathbf{v}) = \frac{\mathbf{v}^\top \mathbf{A} \mathbf{v}}{\mathbf{v}^\top \mathbf{v}}$

Note that if $\mathbf{v}$ is an eigenvector with eigenvalue λ then the Rayleigh coefficient indeed gives us λ. We can find the smallest and largest eigenvalue by minimizing and maximizing this coefficient. But note that if we apply this to the symmetric matrix $\mathbf{X}^\top \mathbf{X} + \lambda' \mathbf{I}$ then for any vector $\mathbf{v}$ we have $\frac{\mathbf{v}^\top(\mathbf{X}^\top \mathbf{X} + \lambda' \mathbf{I}) \mathbf{v}}{\mathbf{v}^\top \mathbf{v}} \geq \frac{\lambda' \mathbf{v}^\top \mathbf{v}}{\mathbf{v}^\top \mathbf{v}} = \lambda'$

L_1-Regularization: The Lasso

- L_1-norm in combination with the MSE cost function, this is known as the Lasso:
$\min_{\mathbf{w}} \frac{1}{2N} \sum_{n=1}^N \left[y_n - \mathbf{x}_n^\top \mathbf{w}\right]^2 + \lambda \|\mathbf{w}\|_1$
where $\|\mathbf{w}\|_1 := \sum_i |w_i|$
- For L_1-regularization the optimum solution is likely going to be sparse (only has few non-zero components) compared to the case where we use L_2-regularization. Sparsity is desirable, since it leads to a "simple" model.

Geometric Interpretation

- Assume that $\mathbf{X}^\top \mathbf{X}$ is invertible. We claim that in this case the set $\left\{\mathbf{w} : \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = \alpha\right\}$ is an ellipsoid and this ellipsoid simply scales around its origin as we change α.
- First look at $\alpha = \|\mathbf{X}\mathbf{w}\|^2 = \mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w}$. This is a quadratic form. Let $\mathbf{A} = \mathbf{X}^\top \mathbf{X}$. Note that $\mathbf{A}$ is a symmetric matrix and by assumption it has full rank. If $\mathbf{A}$ is a diagonal matrix with strictly positive elements a_i along the diagonal then this describes the equation $\sum_i a_i \mathbf{w}_i^2 = \alpha$ which is indeed the equation for an ellipsoid. In the general case, $\mathbf{A}$ can be written as (using the SVD) $\mathbf{A} = \mathbf{U}\mathbf{B}\mathbf{U}^T$, where $\mathbf{B}$ is a diagonal matrix with strictly positive entries. This then corresponds to an ellipsoid with rotated axes. If we now look at $\alpha = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$, where $\mathbf{y}$ is in the column space of $\mathbf{X}$ then we can write it as $\alpha = \|\mathbf{X}(\mathbf{w}_0 - \mathbf{w})\|^2$ for a suitable chosen $\mathbf{w}_0$ and so this corresponds to a shifted ellipsoid. Finally, for the general case, write $\mathbf{y}$ as $\mathbf{y} = \mathbf{y}_\| + \mathbf{y}_\perp$, where $\mathbf{y}_\|$ is the component of $\mathbf{y}$ that lies in the subspace spanned by the columns of $\mathbf{X}$ and $\mathbf{y}_\perp$ is the component that is orthogonal. In this case

$$\begin{aligned}
\alpha &= \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 \\
&= \left\|\mathbf{y}_\| + \mathbf{y}_\perp - \mathbf{X}\mathbf{w}\right\|^2 \\
&= \|\mathbf{y}_\perp\|^2 + \left\|\mathbf{y}_\| - \mathbf{X}\mathbf{w}\right\|^2 \\
&= \|\mathbf{y}_\perp\|^2 + \|\mathbf{X}(\mathbf{w}_0 - \mathbf{w})\|^2 .
\end{aligned}$$

Hence this is then equivalent to the equation $\|\mathbf{X}(\mathbf{w}_0 - \mathbf{w})\|^2 = \alpha - \|\mathbf{y}_\perp\|^2$, proving the claim. From this we also see that if $\mathbf{X}^\top \mathbf{X}$ is not full rank then what we get is not an ellipsoid but a cylinder with an ellipsoidal cross-subsection.

Ridge Regression as MAP estimator

- Least squares as MLE:

$$\begin{aligned}
\mathbf{w}_{\text{lse}} &\overset{(a)}{=} \arg\min_{\mathbf{w}} -\log p(\mathbf{y}, \mathbf{X} \mid \mathbf{w}) \\
&\overset{(b)}{=} \arg\min_{\mathbf{w}} -\log p(\mathbf{X} \mid \mathbf{w}) p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) \\
&\overset{(c)}{=} \arg\min_{\mathbf{w}} -\log p(\mathbf{X}) p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) \\
&\overset{(d)}{=} \arg\min_{\mathbf{w}} \quad -\log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) \\
&\overset{(e)}{=} \arg\min_{\mathbf{w}} \quad -\log\left[\prod_{n=1}^N p(y_n \mid \mathbf{x}_n, \mathbf{w})\right] \\
&\overset{(f)}{=} \arg\min_{\mathbf{w}} -\log\left[\prod_{n=1}^N \mathcal{N}\left(y_n \mid \mathbf{x}_n^\top \mathbf{w}, \sigma^2\right)\right] \\
&= \arg\min_{\mathbf{w}} -\log\left[\prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}\left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2}\right] \\
&= \arg\min_{\mathbf{w}} -N \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) \\
&\quad + \sum_{n=1}^N \frac{1}{2\sigma^2}\left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2 \\
&= \arg\min_{\mathbf{w}} \frac{1}{2\sigma^2} \sum_{n=1}^N \left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2
\end{aligned}$$

- Ridge as MAP:

$$\begin{aligned}
\mathbf{w}_{\text{ridge}} &= \arg\min_{\mathbf{w}} \quad -\log p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}) \\
&\overset{(a)}{=} \arg\min_{\mathbf{w}} -\log \frac{p(\mathbf{y}, \mathbf{X} \mid \mathbf{w}) p(\mathbf{w})}{p(\mathbf{y}, \mathbf{X})} \\
&\overset{(b)}{=} \arg\min_{\mathbf{w}} -\log p(\mathbf{y}, \mathbf{X} \mid \mathbf{w}) p(\mathbf{w}) \\
&\overset{(c)}{=} \arg\min_{\mathbf{w}} \quad -\log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) p(\mathbf{w}) \\
&= \arg\min_{\mathbf{w}} -\log\left[p(\mathbf{w}) \prod_{n=1}^N p(y_n \mid \mathbf{x}_n, \mathbf{w})\right] \\
&= \arg\min_{\mathbf{w}} -\log \mathcal{N}\left(\mathbf{w} \mid 0, \frac{1}{\lambda}\mathbf{I}\right) \\
&\quad \times \prod_{n=1}^N \mathcal{N}\left(y_n \mid \mathbf{x}_n^\top \mathbf{w}, \sigma^2\right) \\
&= \arg\min_{\mathbf{w}} -\log \frac{1}{(2\pi\frac{1}{\lambda})^{D/2}} e^{-\frac{\lambda}{2}\|\mathbf{w}\|^2} \\
&\quad \times \prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}\left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2} \\
&= \arg\min_{\mathbf{w}} \sum_{n=1}^N \frac{1}{2\sigma^2}\left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2
\end{aligned}$$

- Ridge regression has a very similar interpretation.
- We start with the posterior $p(\mathbf{w} \mid \mathbf{X}, \mathbf{y})$ and chose that parameter $\mathbf{w}$ that maximizes this posterior. Hence this is called the maximum-a-posteriori (MAP) estimate.
- As before, we take the log and add a minus sign and minimize instead.
- In order to compute the posterior we use Bayes law and we assume that the components of the weight vector are iid Gaussians with mean zero and variance $\frac{1}{\lambda}$.
In step (a) we used Bayes' law. In step (b) and (c) we eliminated quantities that do not depend on $\mathbf{w}$.

Generalization, validation?

What is the model selection problem?

- Ridge regression: $w_\lambda = \arg\min_w \frac{1}{2N} \sum_{n=1}^N \left(y_n - x_n^\top w\right)^2 + \lambda \|w\|_2^2$
- λ can be tuned to control the model complexity (to reduce overfitting)

Hyperparameter

- In practice: $(\lambda_1, \cdots, \lambda_k) \longrightarrow$ Algorithm $\longrightarrow (w_1, \cdots, w_k)$
- Which λ should we use?
- Polynomial feature expansion: $(x_{(1)}, x_{(2)}) \overset{\phi}{\rightarrow}$
$(x_{(1)}, x_{(2)}, x_{(1)}^2 + x_{(2)}^2)$
$(x_{(1)}, x_{(2)}, 5x_{(1)}^2 + 2x_{(2)}^2, x_{(2)}^3 + 2x_{(1)})$
- Enrich the model complexity, by augmenting the feature vector x.
- Here the degree d is the hyperparameter

We are facing the same problem: how do we choose these hyperparameters?

Model selection for neural networks

Algorithms?
SGD
Adam
Which step-size?
Which batch-size?
Which momentum?

Architectures?

FullyConnected
ConvNet
ResNet
Transformer
Which width?
Which depth?
Batch normalization?

Regularizations?
Weight decay?
Early stopping?
Data augmentations?

Probabilistic Setup

Data Model:
Unknown distribution $\mathscr{D}$ with range $\mathscr{X} \times \mathscr{Y}$
We see a dataset S of independent samples from $\mathscr{D}$:

$$S = \{(x_n, y_n)\}_{n=1}^{N} \sim \mathscr{D} \quad \text{i.i.d.}$$

Learning Algorithm:
Ridge regression: gradient descent or least-squares estimator
Can add a subscript $f_{S,\lambda}$ to indicate the model dependency

Generalization Error: how accurate is f at predicting?

We compute the expected error over all samples drawn from distribution $\mathscr{D}$:

$$L_{\mathscr{D}}(f) = \mathbb{E}_{(x,y)\sim\mathscr{D}}[\ell(y, f(x))]$$

where $\ell(\cdot,\cdot)$ is the loss function

- Ex: $\ell(y, y') = \frac{1}{2}(y - y')^2$, logistic loss, hinge loss

The quantity $L_{\mathscr{D}}(f)$ has many names: $\begin{cases}\text{True}\\ \text{Expected}\\ \text{Generalization}\end{cases}$ $\begin{cases}\text{Risk}\\ \text{Error}\\ \text{Loss}\end{cases}$
This is the quantity we are fundamentally interested in

Generalization Error: how accurate is f at predicting?

We compute the expected error over all samples drawn from distribution $\mathscr{D}$:

$$L_{\mathscr{D}}(f) = \mathbb{E}_{(x,y)\sim\mathscr{D}}[\ell(y, f(x))]$$

where $\ell(\cdot,\cdot)$ is the loss function

- Ex: $\ell(y, y') = \frac{1}{2}(y - y')^2$, logistic loss, hinge loss

The quantity $L_{\mathscr{D}}(f)$ has many names: $\begin{cases}\text{True}\\ \text{Expected}\\ \text{Generalization}\end{cases}$ $\begin{cases}\text{Risk}\\ \text{Error}\\ \text{Loss}\end{cases}$
This is the quantity we are fundamentally interested in

Empirical Error: what we can compute

We can approximate the true error by averaging the loss function over the dataset

$$L_S(f) = \frac{1}{|S|} \sum_{(x_n, y_n)\in S} \ell(y_n, f(x_n))$$

Also called: empirical risk/error/loss
Δ The samples are random thus $L_S(f)$ is a random variable It is an unbiased estimator of the true error
Law of large number: $L_S(f) \underset{|S|\to\infty}{\rightarrow} L_{\mathscr{D}}(f)$ but fluctuations!
Generalization gap: $|L_{\mathscr{D}}(f) - L_S(f)|$

Training error: what we are minimizing

Δ the prediction function f_S is itself a function of the data S
When the model has been trained on the same data it is applied to, the empirical error is called the training error:

$$L_S(f_S) = \frac{1}{|S|} \sum_{(x_n, y_n)\in S} \ell(y_n, f_S(x_n))$$

This is the objective function you are minimizing to find the predictor It might not be representative of the error we see on "fresh" samples
The reason that $L_S(f_S)$ might not be close to $L_{\mathscr{D}}(f_S)$ is of course overfitting

Splitting the data

Problem: Validating model on the same data we trained it on Fix: Split the data into an independent training and test set:

$$S = S_{\text{train}} \cup S_{\text{test}}$$

1. We learn the function $f_{S_{\text{train}}}$ using the train set
2. We validate it computing the error on the test set

$$L_{S_{\text{test}}}(f_{S_{\text{train}}}) = \frac{1}{|S_{\text{test}}|} \sum_{(y_n, x_n)\in S_{\text{test}}} \ell(y_n, f_{S_{\text{train}}}(x_n))$$

$\Rightarrow$ Since S_{test} and S_{train} are independent: $L_{S_{\text{test}}}(f_{S_{\text{train}}}) \approx L_{\mathscr{D}}(f_{\text{train}})$

Splitting the data

Problem: Validating model on the same data we trained it on
Fix: Split the data into an independent training and test set:

$$S = S_{\text{train}} \cup S_{\text{test}}$$

1. We learn the function $f_{S_{\text{train}}}$ using the train set
2. We validate it computing the error on the test set

$$L_{S_{\text{test}}}(f_{S_{\text{train}}}) = \frac{1}{|S_{\text{test}}|} \sum_{(y_n, x_n)\in S_{\text{test}}} \ell(y_n, f_{S_{\text{train}}}(x_n))$$

$\Rightarrow$ Since S_{test} and S_{train} are independent: $L_{S_{\text{test}}}(f_{S_{\text{train}}}) \approx L_{\mathscr{D}}(f_{S_{\text{train}}})$
Δ We have less data both for the learning and the validation tasks (tradeoff)

Generalization gap: How far is the test from the true error?

Claim: given a model f and a test set $S_{\text{test}} \sim \mathscr{D}$ i.i.d. (not used to learn f) and a loss $\ell(\cdot,\cdot) \in [a,b]$:

$$\mathbb{P}\left[\underbrace{|L_{\mathscr{D}}(f) - L_{S_{\text{test}}}(f)|}_{\text{Generalization Gap}} \geq \sqrt{\frac{(b-a)^2\ln(2/\delta)}{2|S_{\text{test}}|}}\right] \leq \delta$$

The error decreases as $\mathcal{O}\left(1/\sqrt{|S_{\text{test}}|}\right)$ with the number of test points High probability bound: δ is only in the ln
$\rightarrow$ The more data points we have, the more confident we are that the empirical loss we measure is close to the true loss

Why do you care?

- Given a predictor f and a dataset S you can control the expected risk:

$$\mathbb{P}(\underbrace{L_{\mathscr{D}}(f)}_{\text{not computable}} \geq \underbrace{L_S(f)}_{\text{Computable}} + \underbrace{\sqrt{\frac{(a-b)^2\ln(2/\delta)}{2|S_{\text{test}}|}}}_{\text{deviation}}) \leq \delta$$

- Given a dataset S

1. Split: $S = S_{\text{train}} \cup S_{\text{test}}$
2. Train: $\mathscr{A}(S_{\text{train}}) = f_{S_{\text{train}}}$
3. Validate:

$$\mathbb{P}\left(L_{\mathscr{D}}(f_{S_{\text{train}}}) \geq L_{S_{\text{test}}}(f_{S_{\text{train}}}) + \sqrt{\frac{(a-b)^2\ln(2/\delta)}{2|S_{\text{test}}|}}\right) \leq \delta$$

$\Rightarrow$ We can obtain a probabilistic upper bound on the expected risk

The proof relies only on concentration inequalities

Since $(x_n, y_n) \in S_{\text{test}}$ are chosen independently, the associated losses $\Theta_n = \ell(y_n, f(x_n)) \in [a,b]$ given a fixed model f, are also i.i.d. random variables
Empirical loss: $\frac{1}{N}\sum_{n=1}^{N}\Theta_n = \frac{1}{N}\sum_{n=1}^{N}\ell(y_n, f(x_n)) = L_{S_{\text{test}}}(f)$
True loss: $\mathbb{E}[\Theta_n] = \mathbb{E}[\ell(y_n, f(x_n))] = L_{\mathscr{D}}(f)$
What is the chance that the empirical loss $L_{S_{\text{test}}}(f)$ deviates from the true loss by more than a given constant?

- classically addressed using concentration inequalities

Hoeffding inequality: a simple concentration bound

Claim: Let $\Theta_1, \ldots, \Theta_N$ be a sequence of i.i.d. random variables with mean $\mathbb{E}[\Theta]$ and range $[a,b]$

$$\mathbb{P}\left[\left|\frac{1}{N}\sum_{n=1}^{N}\Theta_n - \mathbb{E}[\Theta]\right| \geq \varepsilon\right] \leq 2e^{-2N\varepsilon^2/(b-a)^2} \text{ for an}$$

Concentration bound: the empirical mean is concentrated around its mean
A. Use it with $\Theta_n = \ell(y_n, f(x_n))$
B. Equating $\delta = 2e^{-2|S_{\text{test}}|\varepsilon^2/(b-a)^2}$ we get $\varepsilon = \sqrt{\frac{(b-a)^2\ln(2/\delta)}{2|S_{\text{test}}|}}$ $\square$

Model Selection: pick the best model

Goal: select the hyperparameters of our model (λ for the ridge regression) We have a set of values $\{\lambda_k\}_{k=1}^{K}$. Which one should we choose?

1. Split the data into $S = S_{\text{train}} \cup S_{\text{test}}$, generated independently from $\mathscr{D}$
2. Run the learning algorithm K times on the same training set S_{train} to compute the K prediction functions $f_{S_{\text{train}},\lambda_k}$
3. For each prediction function, compute the test error $L_{S_{\text{test}}}(f_{S_{\text{train}},\lambda_k})$

We then choose the value of the parameter λ giving the smallest test error

Examples

Ridge regression
Degree in case of a polynomial feature expansion

Does model selection work?

Two questions:

- How do we know that the best function $f_{S_{\text{train},\lambda}}$ is a good approximation of the best model within our function class?

- How do we know that $L_{S_{\text{test}}}\left(f_{S_{\text{train}},\lambda_k}\right) \approx L_{\mathscr{D}}\left(f_{S_{\text{train}},\lambda_k}\right)$?

We have discussed it for a single model What about several models?
I.e., what is the justification that the min is actually good?

How far is each of the K test errors $L_{S_{\text{test}}}(f_k)$ from the true $L_{\mathscr{D}}(f_k)$?

Claim: we can bound the maximum deviation for all K candidates, by

$$\mathbb{P}\left[\max_k \left|L_{\mathscr{D}}(f_k) - L_{S_{\text{test}}}(f_k)\right| \geq \sqrt{\frac{(b-a)^2 \ln(2K/\delta)}{2\left|S_{\text{test}}\right|}}\right] \leq \delta$$

- The error decreases as $\mathcal{O}\left(1/\sqrt{|S_{\text{test}}|}\right)$ with the number test points
- When testing K hyper-parameters, the error only goes up by $\sqrt{\ln(K)}$

$\Rightarrow$ So we can test many different models without incurring a large penalty

- It can be extended to infinitely many models

Proof: A simple union bound

The proof of this statement follows the proof of the special case $K=1$

$$\begin{aligned}\mathbb{P}\left[\max_k \left|L_{\mathscr{D}}(f_k) - L_{S_{\text{test}}}(f_k)\right| \geq \varepsilon\right] &= \mathbb{P}\left[\cup_k \left\{\left|L_{\mathscr{D}}(f_k) - L_{S_{\text{test}}}(f_k)\right| \geq \varepsilon\right\}\right] \\ &\leq \sum_k \mathbb{P}\left[\left|L_{\mathscr{D}}(f_k) - L_{S_{\text{test}}}(f_k)\right| \geq \varepsilon\right] \\ &\leq 2Ke^{-2N\varepsilon^2/(b-a)^2}\end{aligned}$$

Hence, equating $\delta = 2Ke^{-2N\varepsilon^2/(b-a)^2}$, we get $\varepsilon = \sqrt{\frac{(b-a)^2\ln(2K/\delta)}{2N}}$ as stated
If we choose the "best" function according to the empirical risk then its true risk is not too far away from the true risk of the optimal choice
Let $k^* = \operatorname{argmin}_k L_{\mathscr{D}}(f_k)$ and $\hat{k} = \operatorname{argmin}_k L_{S_{\text{test}}}(f_k)$ then

$$\mathbb{P}\left[L_{\mathscr{D}}(f_{\hat{k}}) \geq L_{\mathscr{D}}(f_{k^*}) + 2\sqrt{\frac{(b-a)^2\ln(2K/\delta)}{2\left|S_{\text{test}}\right|}}\right] \leq \delta$$

Function with
Function with
the smallest empirical risk the smallest true risk

If we choose the "best" function according to the empirical risk then its true risk is not too far away from the true risk of the optimal choice
Let $k^* = \operatorname{argmin}_k L_{\mathscr{D}}(f_k)$ and $\hat{k} = \operatorname{argmin}_k L_{S_{\text{test}}}(f_k)$ then

$$\mathbb{P}\left[L_{\mathscr{D}}(f_{\hat{k}}) \geq L_{\mathscr{D}}(f_{k^*}) + 2\sqrt{\frac{(b-a)^2\ln(2K/\delta)}{2\left|S_{\text{test}}\right|}}\right] \leq \delta$$

Function with
Function with

Cross-Validation

- Splitting the data once into two parts (one for training and one for testing) is not the most efficient way to use the data
- Cross-validation is a better way

K-fold Cross-Validation

1. Randomly partition the data into K groups

3. Average the K results

- We have used all data for training, and all data for testing, and used each data point the same number of times
- Cross-validation returns an estimate of the generalization-error and its variance

Do we still have some time?

Hoeffding's inequality:
Let $\Theta_1, \ldots, \Theta_N$ be a sequence of i.i.d. random variables with mean $\mathbb{E}[\Theta]$ and range $[a,b]$. Then, for any $\varepsilon > 0$

$$\mathbb{P}\left[\left|\frac{1}{N}\sum_{n=1}^{N}\Theta_n - \mathbb{E}[\Theta]\right| \geq \varepsilon\right] \leq 2e^{-2N\varepsilon^2/(b-a)^2}$$

Proof (I)

- We equivalently assume that $\mathbb{E}[\Theta] = 0$ and that $\Theta_n \in [a,b]$
- We will only show that

$$\mathbb{P}\left\{\frac{1}{N}\sum_{n=1}^{N}\Theta_n \geq \varepsilon\right\} \leq e^{-2N\varepsilon^2/(b-a)^2}$$

This, together with the equivalent bound

$$\mathbb{P}\left\{\frac{1}{N}\sum_{n=1}^{N}\Theta_n \leq -\varepsilon\right\} \leq e^{-2N\varepsilon^2/(b-a)^2}$$

will prove the claim

Proof (II)

For any $s \geq 0$, $\mathbb{P}\left\{\frac{1}{N}\sum_{n=1}^{N}\Theta_n \geq \varepsilon\right\} = \mathbb{P}\left\{s\frac{1}{N}\sum_{n=1}^{N}\Theta_n \geq s\varepsilon\right\}$

$$\begin{aligned}&= \mathbb{P}\left\{e^{s\frac{1}{N}\sum_{n=1}^{N}\Theta_n} \geq e^{s\varepsilon}\right\} \\ &\leq \mathbb{E}\left[e^{s\frac{1}{N}\sum_{n=1}^{N}\Theta_n}\right]e^{-s\varepsilon} \qquad \text{(Markov inequality)} \\ &= \prod_{n=1}^{N}\mathbb{E}\left[e^{\frac{s\Theta_n}{N}}\right]e^{-s\varepsilon} \qquad \text{(the r.v } \Theta_n \text{ are independent)} \\ &= \mathbb{E}\left[e^{\frac{s\Theta}{N}}\right]^N e^{-s\varepsilon} \qquad \text{(the r.v } \Theta_n \text{ are i.d.)} \\ &\leq e^{s^2(b-a)^2/(8N)}e^{-s\varepsilon} \qquad \text{(Hoeffding lemma)}\end{aligned}$$

Proof (III)

What do we do now? We have for any $s \geq 0$

$$\mathbb{P}\left\{\frac{1}{N}\sum_{n=1}^{N}\Theta_n \geq \varepsilon\right\} \leq e^{s^2(b-a)^2/(8N)}e^{-s\varepsilon}$$

In particular for the minimum value obtained for $s = \frac{4N\varepsilon}{(b-a)^2}$

$$\mathbb{P}\left\{\frac{1}{N}\sum_{n=1}^{N}\Theta_n \geq \varepsilon\right\} \leq e^{-2N\varepsilon^2/(b-a)^2}$$

Hoeffding lemma

For any random variable X, with $\mathbb{E}[X] = 0$ and $X \in [a,b]$ we have

$$\mathbb{E}\left[e^{sX}\right] \leq e^{\frac{1}{8}s^2(b-a)^2} \text{ for any } s \geq 0$$

Proof outline:
Consider the convex function $s \mapsto e^{sx}$. In the range $[a,b]$ it is upper bounded by the chord

$$e^{sx} \leq \frac{x-a}{b-a}e^{sb} + \frac{b-x}{b-a}e^{sa}$$

Taking the expectation and recalling that $\mathbb{E}[X] = 0$, we get

$$\mathbb{E}\left[e^{sX}\right] \leq \frac{b}{b-a}e^{sa} - \frac{a}{b-a}e^{sb} \leq e^{s^2(b-a)^2/8}$$

Last time

How can we judge if a given predictor is good?
How to select the best models of a family?
$\rightarrow$ Bound the difference between the true and empirical risks
$\rightarrow$ Split data into train and test sets (learn with the train and test on the test)
Motivation: Hyperparameters search (which often control the complexity)
But we haven't investigated the role of the complexity of the class

Today

How does the risk behave as a function of the complexity of the model class?
$\Rightarrow$ Bias-Variance tradeoff
It will help us to decide how complex and rich we should make our model
Before: quantitative
Now: qualitative

A small experiment: 1D-regression

True function

A small experiment: 1D-regression

Sampled training set
Linear regression using polynomial feature expansion $\left(x, x^2, x^3, \cdots, x^d\right)$ The maximum degree d measures the complexity of the class
$\Rightarrow$ How far should you go?

Simple model: bad fit

Training fit (degree 1)
No linear function would be a good predictor. The model class is not rich enough

Complex model: good fit?

High degree polynomial will be a good fit. But?

But there is randomness in the data

Sampled training set
We have observed one particular S_{train} but we could have observed several others!

But there is randomness in the data

Sampled training set
Even if we keep the same $(x_1, \cdots, x_n)$, we have variability in the observed $(y_1, \cdots, y_n)$

Thus there is randomness in the predictions

Training fit (degree 1)
Moving a single observation will cause only a small shift in the position of the line
Underfitting Training fit (degree 12)
Overfitting
Simple models are less sensitive

Simple models have large bias but low variance

The average of the predictions f_S does not fit well the data: large bias The variance of the predictions f_S as a function of S is small: small variance

Complex models have low bias but high variance

The average of the predictions f_S fits well the data: small bias

The variance of the predictions f_S as a function of S is large: large variance

We need to balance bias & variance correctly

Data model: output perturbed by some noise

We consider the square loss and will provide a decomposition of the true error

Error Decomposition

We are interested in how the expected error of f_S :

$$\mathbb{E}_{(x,y)\sim D}\left[(y-f_S(x))^2\right]$$

behaves as a function of the train set S and model class complexity

Error Decomposition

The decomposition will hold true at every single point x. Therefore, to simplify, we consider the expected error of f_S for a fixed element x_0 :

$$L\left(f_S\right)=\mathbb{E}_{\varepsilon\sim\mathscr{D}_\varepsilon}\left[\left(f\left(x_0\right)+\varepsilon-f_S\left(x_0\right)\right)^2\right]$$

This is a random variable. The randomness comes for the train set S

We run the experiment many times

We are interested in the average and the variance of the predictions $(f_{S_1},\cdots,f_{S_k})$ over these multiple runs

A decomposition in three terms

We are interested in the expectation of the true risk over the training set S

$$\begin{aligned}\mathbb{E}_{S\sim\mathscr{D}}\left[L\left(f_S\right)\right]&=\mathbb{E}_{S\sim\mathscr{D}}\left[\mathbb{E}_{\varepsilon\sim D_\varepsilon}\left[\left(f\left(x_0\right)+\varepsilon-f_S\left(x_0\right)\right)^2\right]\right]\\&=\mathbb{E}_{S\sim\mathscr{D},\varepsilon\sim\mathscr{D}_\varepsilon}\left[\left(f\left(x_0\right)+\varepsilon-f_S\left(x_0\right)\right)^2\right]\end{aligned}$$

We will decompose this quantity in three non-negative terms and will interpret each of these terms

First we expand the square:

$$\begin{aligned}&\mathbb{E}_{S\sim\mathscr{D},\varepsilon\sim\mathscr{D}_\varepsilon}\left[\left(f\left(x_0\right)+\varepsilon-f_S\left(x_0\right)\right)^2\right]=\mathbb{E}_{\varepsilon\sim\mathscr{D}_\varepsilon}\left[\varepsilon^2\right]\\&+2\mathbb{E}_{S\sim\mathscr{D},\varepsilon\sim\mathscr{D}_\varepsilon}\left[\varepsilon\left(f\left(x_0\right)-f_S\left(x_0\right)\right)\right]\\&+\mathbb{E}_{S\sim\mathscr{D}}\left[\left(f\left(x_0\right)-f_S\left(x_0\right)\right)^2\right]\end{aligned}$$

Using that $\mathbb{E}_{\varepsilon\sim\mathscr{D}}[\varepsilon]=0$ and εS :

- $\mathbb{E}_{\varepsilon\sim\mathscr{D}_\varepsilon}\left[\varepsilon^2\right]=\mathrm{Var}_{\varepsilon\sim\mathscr{D}_\varepsilon}[\varepsilon]$
- $\mathbb{E}_{S\sim\mathscr{D},\varepsilon\sim\mathscr{D}_\varepsilon}\left[\varepsilon\left(f\left(x_0\right)-f_S\left(x_0\right)\right)\right]=\mathbb{E}_{\varepsilon\sim\mathscr{D}\varepsilon}[\varepsilon]\times\mathbb{E}_{S\sim\mathscr{D}}\left[f\left(x_0\right)-f_S\left(x_0\right)\right]=0$

Therefore

$$\mathbb{E}_{S\sim\mathscr{D},\varepsilon\sim\mathscr{D}_\varepsilon}\left[\left(f\left(x_0\right)+\varepsilon-f_S\left(x_0\right)\right)^2\right]=\mathrm{Var}_{\varepsilon\sim D_\varepsilon}[\varepsilon]+\mathbb{E}_{S\sim\mathscr{D}}\left[\left(f\left(x_0\right)-f_S\left(x_0\right)\right)^2\right]$$

Trick: we add and subtract the constant term $\mathbb{E}_{S'\sim D}\left[f_{S'}\left(x_0\right)\right]$, where S' is a second training set independent from S

$$\begin{aligned}\mathbb{E}_{S\sim\mathscr{D}}\left[\left(f\left(x_0\right)-f_S\left(x_0\right)\right)^2\right]=&\mathbb{E}_{S\sim\mathscr{D}}\left[\left(f\left(x_0\right)-\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]+\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]-f_S\left(x_0\right)\right)^2\right]\\=&\mathbb{E}_{S\sim\mathscr{D}}\left[\left(f\left(x_0\right)-\mathbb{E}_{S'\sim\mathcal{D}}\left[f_{S'}\left(x_0\right)\right]\right)^2+\left(\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]-f_S\left(x_0\right)\right)^2\right.\\&\left.+2\left(f\left(x_0\right)-\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]\right)\left(\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]-f_S\left(x_0\right)\right)\right]\end{aligned}$$

Cross-term:

$$\begin{aligned}&\mathbb{E}_{S\sim\mathscr{D}}\left[\left(f\left(x_0\right)-\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]\right)\cdot\left(\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]-f_S\left(x_0\right)\right)\right]\\&=\left(f\left(x_0\right)-\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]\right)\cdot\mathbb{E}_{S\sim\mathscr{D}}\left[\left(\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]-f_S\left(x_0\right)\right)\right]\\&=\left(f\left(x_0\right)-\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]\right)\cdot\left(\mathbb{E}_{S'\sim\mathcal{D}}\left[f_{S'}\left(x_0\right)\right]-\mathbb{E}_{S\sim\mathscr{D}}\left[f_S\left(x_0\right)\right]\right)=0.\end{aligned}$$

$$\mathbb{E}_{S\sim\mathscr{D}}\left[\left(f\left(x_0\right)-f_S\left(x_0\right)\right)^2\right]=\left(f\left(x_0\right)-\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]\right)^2+\mathbb{E}_{S\sim\mathscr{D}}\left[\left(\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]-f_S\left(x_0\right)\right)^2\right]$$

Bias-Variance Decomposition

We obtain the following decomposition into three positive terms:

$\mathbb{E}_{S\sim\mathscr{D},\varepsilon\sim\mathscr{D}_\varepsilon}\left[\left(f\left(x_0\right)+\varepsilon-f_S\left(x_0\right)\right)^2\right]=\mathrm{Var}_{\varepsilon\sim\mathscr{D}_\varepsilon}[\varepsilon]\longleftarrow$ Noise variance

Bias $\longrightarrow+\left(f\left(x_0\right)-\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]\right)^2$

Variance $\longrightarrow+\mathbb{E}_{S\sim\mathscr{D}}\left[\left(f_S\left(x_0\right)-\mathbb{E}_{S'\sim\mathscr{D}}\left[f_{S'}\left(x_0\right)\right]\right)^2\right]$

each of which always provides a lower bound of the true error

⇒ To minimize the true error, we must choose a method that achieves low bias and low variance simultaneously

Noise: a strict lower bound on the achievable error

- It is not possible to go below the noise level
- Even if we know the true model f, we still suffer from the noise: $L(f)=\mathbb{E}\left[\varepsilon^2\right]$
- It is not possible to predict the noise from the data since they are independent

Bias: $\left(f\left(x_0\right)-\mathbb{E}_{S\sim\mathscr{D}}\left[f_S\left(x_0\right)\right]\right)^2$

- Squared of the difference between the actual value $f\left(x_0\right)$ and the expected prediction
- It measures how far off in general the models' predictions are from the correct value
- If model complexity is low, bias is typically high
- If model complexity is high, bias Is typically low

Bias: $\left(f\left(x_0\right)-\mathbb{E}_{S\sim\mathscr{D}}\left[f_S\left(x_0\right)\right]\right)^2$

- Squared of the difference between the actual value $f\left(x_0\right)$ and the expected prediction
- It measures how far off in general the models' predictions are from the correct value
- If model complexity is low, bias is typically high
- If model complexity is high, bias is typically low

Variance: $\mathbb{E}_{S\sim\mathscr{D}}\left[\left(f_S\left(x_0\right)-\mathbb{E}_{S\sim\mathscr{D}}\left[f_S\left(x_0\right)\right]\right)^2\right]$

- Variance of the prediction function
- It measures the variability of predictions at a given point across different training set realizations
- If we consider complex models, small variations in the training set can lead to significant changes in the predictions

Variance: $\mathbb{E}_{S\sim\mathscr{D}}\left[\left(f_S\left(x_0\right)-\mathbb{E}_{S\sim D}\left[f_S\left(x_0\right)\right]\right)^2\right]$

- Variance of the prediction function
- It measures the variability of predictions at a given point across different training set realizations
- If we consider complex models, small variations in the training set can lead to significant changes in the predictions

Bias Variance tradeoff and U-shape curve

- If model complexity is too low, approximation will be poor (underfitting)
- If model complexity is too high, it may cause issues with variance (overfitting)

⇒ This phenomenon is known as the bias-variance tradeoff

Challenge: Identify a method that ensures both low variance and low bias

Conclusion

Low Variance

Low Bias

High Bias

High Variance

But this depends on the algorithm!

Double descent curve

Reconciling modern machine-learning practice and the classical bias-variance trade-off Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal, PNAS, 2019

Classification

- Linear Decision boundaries: Assume we restrict ourselves to linear decision boundaries (hyperplane):

⇒ Prediction: $f(x)=\operatorname{sign}\left(x^\top w\right)$

- Assume the data are linearly separable, i.e., a separating hyperplane exists

Margin

- Key concept: The margin is the distance from the hyperplane to the closest point

⇒ Take the one with the largest margin!

- Max-margin separating hyperplane: Choose the hyperplane which maximizes the margin Why: If we slightly change the training set, the number of misclassifications will stay low

⇒ It will lead us to support vector machine (SVM) and logistic regression

Formalizing Binary Classification

- Setting: $(X,Y)\sim\mathscr{D}$ with ranges $\mathscr{X},\mathscr{Y}=\{-1,1\}$

- Loss function: (0-1 Loss) $\ell\left(y,y'\right)=1_{y\neq y'}=\begin{cases}1&\text{if } y\neq y'\\0&\text{if } y=y'\end{cases}$

- True risk for the classification: $L_{\mathscr{D}}(f)=\mathbb{E}_{\mathscr{D}}\left[1_{Y\neq f(X)}\right]=\mathbb{P}_{\mathscr{D}}[Y\neq f(X)]$ (resp. classification error = probability of making an error)

- Goal: minimize $L_{\mathscr{D}}(f)$

Bayes classifier

- Def: The classifier $f_*=\arg\min L_{\mathscr{D}}(f)$ is called the Bayes classifier f

- Claim: $f_*(x)=\arg\max_{y\in\{-1,1\}}\mathbb{P}(Y=y\mid X=x)$

- Note: Bayes classifier is an unattainable gold standard, as we never know the underlying data distribution $\mathscr{D}$ in practice

Proof of the Bayes classifier

- Claim 1: $\forall x \in \mathscr{X}, f_*(x) \in \arg\min_{y\in\mathscr{Y}} \mathbb{P}(Y \neq y \mid X = x) \Longrightarrow f_* \in \arg\min_{f:\mathscr{X}\to\mathscr{Y}} L_{\mathscr{D}}(f)$

$L_{\mathscr{D}}(f) = \mathbb{E}_{X,Y}\left[1_{Y\neq f(X)}\right] = \mathbb{E}_X\left[\mathbb{E}_{Y|X}\left[1_{Y\neq f(X)} \mid X\right]\right]$
$= \mathbb{E}_X[\mathbb{P}(Y\neq f(X) \mid X)]$
$\geq \mathbb{E}_X\left[\min_{y\in\mathscr{Y}} \mathbb{P}(Y\neq y \mid X)\right]$
$= \mathbb{E}_X\left[\mathbb{P}(Y\neq f_*(X) \mid X)\right] = \mathbb{E}_{X,Y}\left[1_{Y\neq f_*(X)}\right] = L_{\mathscr{D}}(f_*)$

- Claim 2: $f_*(x) = \arg\min_{y\in\mathscr{Y}} \mathbb{P}(Y\neq y \mid X = x)$
$f_*(x) = \arg\max_{y\in\mathscr{Y}} \mathbb{P}(Y = y \mid X = x) = \arg\min_{y\in\mathscr{Y}} \mathbb{P}(Y \neq y \mid X = x)$

Two classes of classification algorithms

- Non-parametric: approximate the conditional distribution $\mathbb{P}(Y = y \mid X = x)$ via local averaging $\Rightarrow$ KNN
- Parametric: approximate true distribution $\mathscr{D}$ via training data $\Rightarrow$ Minimize the empirical risk on training data (ERM)

Classification by empirical risk minimization

- How: minimize the empirical risk instead of the true risk:

$\min_{f:X\to Y} L_{\text{train}}(f) := \frac{1}{N}\sum_{n=1}^N 1_{f(x_n)\neq y_n} = \frac{1}{N}\sum_{n=1}^N 1_{y_n f(x_n)\leq 0}$

- Problem: L_{train} is not convex:

The set of classifiers is not convex because $\mathcal{Y}$ is discrete

The indicator function 1 is not convex because it is not continuous

Convex relaxation of the classification risk

- Instead of learning $f: \mathscr{X}\to\mathscr{Y}$, learn $g: \mathscr{X}\to\mathbb{R}$ in a convex subset of continuous functions $\mathscr{G}$, and predict with $f(x) = \operatorname{sign}(g(x))$. The problem becomes

$\min_{g\in\mathscr{G}} \frac{1}{N}\sum_{n=1}^N 1_{y_n g(x_n)\leq 0}$

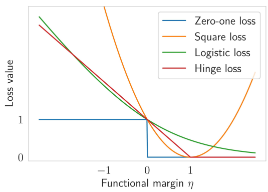
- Replace the indicator function by a convex $\phi: \mathbb{R}\to\mathbb{R}$ and minimize

$\min_{g\in\mathscr{G}} \frac{1}{N}\sum_{n=1}^N \phi(y_n g(x_n))$

ϕ is a function of the functional margin $y_n g(x_n)$

$\Rightarrow$ This is a convex problem!

Remark: possible to bound the zero-one risk $L(f)$ by the ϕ risk



Losses for Classification

- Logistic loss $\to$ logistic regression
- Hinge loss $\to$ max margin classification

Good regressor $\Rightarrow$ good classifier

- Consider $\mathscr{Y} = \{0,1\}$, for all regression functions $\eta: \mathscr{X}\to\mathbb{R}$ we can define a classifier as $\mathscr{X}\to\{0,1\}$

$f_\eta: x \mapsto 1_{\eta(x)\geq 1/2}$

- Claim:

$L_{\mathscr{D}}^{\text{classif}}(f_\eta) - L_{\mathscr{D}}^{\text{classif}}(f^*) \leq 2\sqrt{L_{\mathscr{D}}^{\ell_2}(\eta) - L_{\mathscr{D}}^{\ell_2}(\eta^*)}$

Where $L_{\mathscr{D}}^{\text{classif}}(f_\eta) = \mathbb{E}_{\mathscr{D}}\left[1_{f(X)\neq Y}\right], L_{\mathscr{D}}^{\ell_2}(f) = \mathbb{E}_{\mathscr{D}}\left[(Y - f(X))^2\right]$ and $\eta_* = \arg\min_\eta L_{\mathscr{D}}^{\ell_2}(\eta)$

$\Rightarrow$ If η is good for regression then f_η is good for classification too (converse is not true)

Logistic Regression

- Binary classification : We observe some data $S = \{x_n, y_n\}_{n=1}^N \in \mathscr{X}\times\{0,1\}$

Motivation for logistic regression

Instead of directly modeling the output Y, we can model the probability that Y belongs to a specific class. Map the prediction from $(-\infty, +\infty)$ to $[0,1]$

The logistic function

$\sigma(\eta) := \frac{e^\eta}{1+e^\eta}$

- Properties of the logistic function:

$1 - \sigma(\eta) = \frac{1+e^\eta - e^\eta}{1+e^\eta} = (1+e^\eta)^{-1}$ $\sigma'(\eta) = \frac{e^\eta(1+e^\eta) - e^\eta e^\eta}{(1+e^\eta)^2} = \frac{e^\eta}{(1+e^\eta)^2} = \sigma(\eta)(1-\sigma(\eta))$

Logistic Regression

$p(1 \mid x) := \mathbb{P}(Y = 1 \mid X = x) = \sigma\left(x^\top w + w_0\right)$

$p(0 \mid x) := \mathbb{P}(Y = 0 \mid X = x) = 1 - \sigma\left(x^\top w + w_0\right)$

Logistic regression models the probability that Y belongs to a particular class using the logistic function σ

Label prediction: quantize the probability:

If $p(1 \mid x) \geq 1/2$, you predict the class 1

If $p(1 \mid x) < 1/2$, you predict the class 0

- Interpretation:

Very large $\left|x^\top w + w_0\right|$ corresponds to $p(1 \mid x)$ very close to 0 or 1 (high confidence)

Small $\left|x^\top w + w_0\right|$ corresponds to $p(1 \mid x)$ very close to .5 (low confidence)

Comparison of logistic and linear regression for data with extreme values

- More robust to unbalanced data and extremes.

Geometric Interpretation

- The vector w is orthogonal to the "surface of transition"
- The transition between the two levels happens at the hyperplane $w^\perp = \left\{v, v^\top w = 0\right\}$
- Scaling w makes the transition faster or slower
- Changing w_0 shifts the decision region along the w vector
- The transition happens at the hyperplane $\left\{v, v^\top w + w_0 = 0\right\}$
- Bias term: should consider a shift w_0 as there is no reason for the transition hyperplane to pass through the origin: $p(1 \mid x) = \sigma\left(w^\top x + w_0\right)$ For simplicity, add the constant 1 to the feature vector $x = \begin{pmatrix} x \\ 1 \end{pmatrix}$ It is crucial for allowing to shift the decision region

MLE for logistic regression

- Assumption: The inputs $\mathbf{X}$ do not depend on the parameter w we choose:

$\mathscr{L}(w) = p(\mathbf{y}, \mathbf{X} \mid w) = p(\mathbf{X} \mid w) p(\mathbf{y} \mid \mathbf{X}, w) \underset{\mathbf{X}\perp\perp w}{=} p(\mathbf{X}) p(\mathbf{y} \mid \mathbf{X}, w)$

$$\begin{aligned} p(\mathbf{y} \mid \mathbf{X}, w) &= \Pi_{n=1}^N p(y_n \mid x_n, w) \\ &= \Pi_{n: y_n = 1} p(y_n = 1 \mid x_n, w) \\ &\quad \times \Pi_{n: y_n = 0} p(y_n = 0 \mid x_n, w) \\ &= \Pi_{n=1}^N \sigma\left(x_n^\top w\right)^{y_n}\left[1 - \sigma\left(x_n^\top w\right)\right]^{1-y_n} \end{aligned}$$

The likelihood is proportional to:

$\mathscr{L}(w) \propto \prod_{n=1}^N \sigma\left(x_n^\top w\right)^{y_n}\left[1 - \sigma\left(x_n^\top w\right)\right]^{1-y_n}$

Minimum of the NLL

$$\begin{aligned} & -\log(p(\mathbf{y} \mid \mathbf{X}, w)) \\ &= -\log\left(\prod_{n=1}^N \sigma\left(x_n^\top w\right)^{y_n}\left[1 - \sigma\left(x_n^\top w\right)\right]^{1-y_n}\right) \\ &= -\sum_{n=1}^N y_n \log\sigma\left(x_n^\top w\right) \\ &\quad + (1 - y_n)\log\left(1 - \sigma\left(x_n^\top w\right)\right) \\ &= \sum_{n=1}^N y_n \log\left(\frac{1 - \sigma\left(x_n^\top w\right)}{\sigma\left(x_n^\top w\right)}\right) - \log\left(1 - \sigma\left(x_n^\top w\right)\right) \\ &= \sum_{n=1}^N -y_n x_n^\top w + \log\left(1 + e^{x_n^\top w}\right) \end{aligned}$$

$$\longleftarrow 1 - \sigma(\eta) = \frac{1}{1+e^\eta} \Longrightarrow \frac{1-\sigma(\eta)}{\sigma(\eta)} = e^{-\eta}$$

We obtain the following cost function we will minimize to learn the parameter $w_* = \arg\min L(w)$

$:= \frac{1}{N}\sum_{n=1}^N -y_n x_n^\top w + \log\left(1 + e^{x_n^\top w}\right)$

- If we are considering $y \in \{-1, 1\}$, we will have a different function

A side note on logistic loss

- In logistic regression, the negative log likelihood is equivalent to ERM for the logistic loss (a surrogate for $0-1$ loss, as discussed yesterday)
- Logistic loss for $y \in \{0,1\}$:

$\ell(y, g(x)) = -y g(x) + \log(1 + \exp(g(x)))$

- Logistic loss for $y \in \{-1, 1\}$:

$\ell(y, g(x)) = \log(1 + \exp(-y g(x)))$

- Note: the logistic loss can be applied in modern machine learning as well: $g(x)$ can represent the output of a neural network

Gradient of the negative log likelihood

$\nabla L(w) = \nabla\left[\frac{1}{N}\sum_{n=1}^N \log\left(1 + e^{x_n^\top w}\right) - y_n x_n^\top w\right] = \frac{1}{N}\sum_{n=1}^N \frac{e^{x_n^\top w} x_n}{1+e^{x_n^\top w}} - y_n x_n = \frac{1}{N}\sum_{n=1}^N \left(\sigma\left(x_n^\top w\right) - y_n\right)$

$$\nabla L(w) = \frac{1}{N}\mathbf{X}^\top(\sigma(\mathbf{X}w) - \mathbf{y})$$

- Same gradient as in LS but with σ - No closed form solution to $\nabla L(w) = 0$ - Good news: the cost function L is convex

Convexity of the loss function L

$L(w) = \frac{1}{N}\sum_{n=1}^N -y_n x_n^\top w + \log\left(1 + e^{x_n^\top w}\right)$

is convex in the weight vector w

- Proof: L is obtained through simple convexity preserving operations:

1) Positive combinations of convex functions is convex
2) Composition of a convex and a linear functions is convex
3) A linear function is both convex and concave
4) $\eta \mapsto \log(1 + e^\eta)$ is convex

- Proof of 4: $h(\eta) := \log(1 + e^\eta)$ is cvx

$h'(\eta) = \frac{e^\eta}{1+e^\eta} = \sigma(\eta)$

$h''(\eta) = \sigma'(\eta) = \frac{e^\eta}{(1+e^\eta)^2} \geq 0$

- 2) + 4) $\Rightarrow \log\left(1 + e^{x_n^\top w}\right)$ is convex
- 3) $\Rightarrow -y_n x_n^\top w$ is convex
- 1) $\Rightarrow L(w)$ is convex

Second proof: Hessian of L is psd

- The Hessian $\nabla^2 L$ is the matrix whose entries are the second derivatives $\frac{\partial^2}{\partial w_i \partial w_j} L(w)$

$$\begin{aligned} \nabla^2 L(w) &= \nabla[\nabla L(w)]^\top \\ &= \nabla\left[\frac{1}{N}\sum_{n=1}^N x_n\left(\sigma\left(x_n^\top w\right) - y_n\right)\right]^\top \\ &= \frac{1}{N}\sum_{n=1}^N \nabla\sigma\left(x_n^\top w\right) x_n^\top = \frac{1}{N}\sum_{n=1}^N \sigma\left(x_n^\top w\right) \end{aligned}$$

It can be written under the matrix form:

$\nabla^2 L(w) = \frac{1}{N}\mathbf{X}^\top S\mathbf{X}$

where $S = \operatorname{diag}\left[\sigma\left(x_n^\top w\right)\left(1 - \sigma\left(x_n^\top w\right)\right)\right] \succcurlyeq 0$

$\Rightarrow$ L is convex since $\nabla^2 L(w) \succcurlyeq 0$

How to minimize the convex function L?

GD:

$$\begin{cases} w_0 \in \mathbb{R}^d \\ w_{t+1} = w_t - \frac{\gamma_t}{N}\sum_{n=1}^N \left(\sigma\left(x_n^\top w_t\right) - y_n\right) x_n \end{cases}$$

- can be slow $(\Theta(N \times T))$

SGD:

$$\begin{cases} w_0 \in \mathbb{R}^d \\ w_{t+1} = w_t - \gamma_t \left(\sigma\left(x_{n_t}^\top w_t\right) - y_{n_t}\right) x_{n_t} \end{cases}$$

where $\mathbb{P}[n_t = n] = 1/N$

- is faster but converges slower

Newton's method uses second order information

- Newton's method minimizes the quadratic approximation:

$L(w) \sim L(w_t) + \nabla L(w_t)^\top (w - w_t) + \frac{1}{2}(w - w_t)^\top \nabla^2 L(w_t)(w - w_t) := \phi_t(w)$

$\tilde{w} = \arg\min \phi_t(w) \Rightarrow$

$\nabla L(w_t) + \nabla^2 L(w_t)(\tilde{w} - w_t) = 0$

- Newton's method:

$w_{t+1} = w_t - \gamma_t \nabla^2 L(w_t)^{-1} \nabla L(w_t)$

- The step-size is needed to ensure convergence (damped Newton's method)

- The convergence is typically faster than with gradient descent but the computational complexity is higher (computing Hessian and solving a linear system)

Problem when the data are linearly separable

- Weights go to infinity.

$\inf_w L(w) = 0 = \lim_{\alpha \to \infty} L(\alpha \cdot \bar{w})$

- The inf value is not attained for a finite w

- If we use an optimization algorithm, the weights will go to ∞

- Solution: add a ℓ_2-regularization

- Ridge logistic regression:

$\frac{1}{N}\sum_{n=1}^N -y_n x_n^\top w + \log\left(1 + e^{x_n^\top w}\right) + \frac{\lambda}{2}\|w\|_2^2$

- Optimization perspective: stabilize the optimization process

- Statistical perspective: avoid overfitting

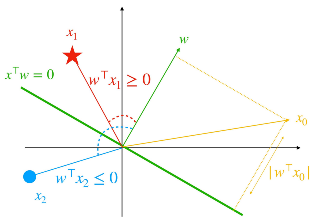
Support Vector Machines

- Vapnik's invention: A training algorithm for optimal margin classifiers

Linear Classifier

- Define a hyperplane as $\{x : w^\top x = 0\}$ where $\|w\| = 1$

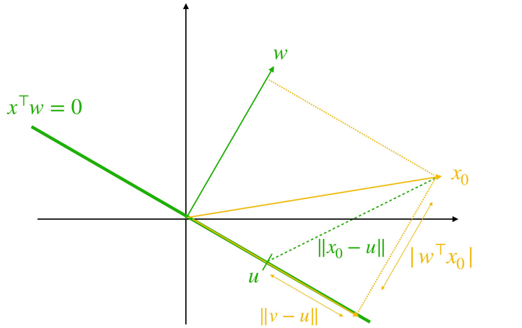
- Prediction: $f(x) = \operatorname{sign}(x^\top w)$



- Claim: The distance between a point x_0 and the hyperplane defined by w is $|w^\top x_0|$

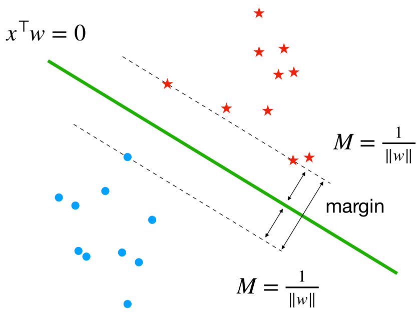
- Proof: The distance between x_0 and the hyperplane is given by $\min_{u: w^\top u = 0} \|x_0 - u\|$

Let $v = x_0 - w^\top x_0 w$ then by the Pythagorean theorem for any u s.t. $w^\top u = 0$



$\|x_0 - u\|^2 = \left(w^\top x_0\right)^2 + \|v - u\|^2 \geq \left(w^\top x_0\right)^2$

Hard-SVM rule: max-margin separating hyperplane



- First assume the dataset $(x_n, y_n)_{n=1}^N$ is linearly separable

- Margin of a hyperplane: $\min_{n \leq N} |w^\top x_n|$

- Max-margin separating hyperplane:

$\max_{w, \|w\|=1} \min_{n \leq N} |w^\top x_n|$ s.t. $\forall n, y_n x_n^\top w \geq 0$

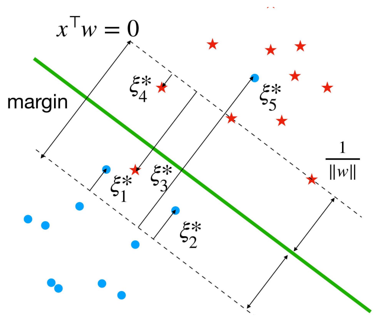
- Equivalent to:

$\max_{M \in \mathbb{R}, w, \|w\|=1} M$ s.t. $\forall n, y_n x_n^\top w \geq M$

- Also equivalent to:

$\min_w \frac{1}{2}\|w\|^2$ such that $\forall n, y_n x_n^\top w \geq 1$

Soft SVM



- A relaxation of the Hard-SVM rule that can be applied even if the training set is not linearly separable

- Idea: Maximize the margin while allowing some constraints to be violated

- Introduce positive slack variables $\xi_1, \cdots, \xi_N$ and replace the constraints with $y_n x_n^\top w \geq 1 - \xi_n$

- Soft SVM:

$$\min_{w, \xi} \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^N \xi_n$$

s.t. $\forall n, y_n x_n^\top w \geq 1 - \xi_n$ and $\xi_n \geq 0$

- Equivalent to:

$\min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^N \left[1 - y_n x_n^\top w\right]_+$

(A hinge loss)

$\xi_i^* = \frac{\xi_i}{\|w\|}$

- Proof: Fix w and consider the minimization over ξ :

1) If $y_n x_n^\top w \geq 1$, then $\xi_n = 0$

2) If $y_n x_n^\top w < 1, \xi_n = 1 - y_n x_n^\top w$

Therefore $\xi_n = \left[1 - y_n x_n^\top w\right]_+$

Classification by risk minimization

- $(X, Y) \sim \mathscr{D}$ with ranges $\mathscr{X}$ and $\mathscr{Y} = \{-1, 1\}$

- Goal: Find a classifier $f : \mathscr{X} \to \mathcal{Y}$ that minimizes the true risk $L(f) = \mathbb{E}_{\mathscr{D}}\left(1_{Y \neq f(X)}\right)$

- How: Through Empirical Risk Minimization (ERM):

$\min_w L_{\text{train}}(w) = \frac{1}{N}\sum_{n=1}^N \phi\left(y_n w^\top x_n\right)$

ϕ represents the loss function of the functional margin $y_n x_n^\top w$

ϕ also serves as a convex surrogate for the $0-1$ loss

Losses for Classification

Examples of margin-based losses $\left(\eta = y x^\top w\right)$:

- Quadratic loss: $\mathrm{MSE}(\eta) = (1 - \eta)^2$
- Logistic loss: $\mathrm{Logistic}(\eta) = \frac{\log(1+\exp(-\eta))}{\log(2)}$
- Hinge loss: $\mathrm{Hinge}(\eta) = [1 - \eta]_+$

Common features: these losses are convex and provide an upper bound for the zero-one loss

Behavioral differences:

- MSE: Penalizes any deviation from 1
- Logistic Loss: Asymmetric cost - a penalty is always incurred.
- Hinge Loss: A penalty is applied if the prediction is incorrect or lacks confidence

Summary

$$\min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^N \left[1 - y_n x_n^\top w\right]_+$$

ERM for the hinge loss with ridge regularization

Interpretation for separable data with small λ :

1. Choose the direction of w such that $w^\perp$ acts as a separating hyperplane
2. Adjust the scale of w to ensure that no point lies with the margin
3. Select the hyperplane with the largest margin

Optimization: How to get w ?

$$\min_w \frac{1}{N}\sum_{n=1}^N \left[1 - y_n x_n^\top w\right]_+ + \frac{\lambda}{2}\|w\|^2$$

Convex (but non-smooth) objective which can be minimized with:

- Subgradient method
- Stochastic Subgradient method

Convex duality

Assume you can define an auxiliary function $G(w, \alpha)$ such that

$$\min_w L(w) = \min_w \max_\alpha G(w, \alpha)$$

Primal problem: $\min_w \max_\alpha G(w, \alpha)$

Dual problem: $\max_\alpha \min_w G(w, \alpha)$

$\Rightarrow$ Sometimes, the dual problem is easier to solve than the primal problem.

Questions:

1. How do we identify a suitable $G(w, \alpha)$?
2. Under what conditions can the min and max be interchanged?
3. When is the dual problem more tractable than the primal problem?

Q1: How do we find a suitable $G(w, \alpha)$?

$$[z]_+ = \max(0, z) = \max_{\alpha \in [0,1]} \alpha z$$

Therefore $\left[1 - y_n x_n^\top w\right]_+ = \max_{\alpha_n \in [0,1]} \alpha_n \left(1 - y_n x\right.$

The SVM problem is equivalent to:

$$\min_w L(w) = \min_w \max_{\alpha \in [0,1]^n} \underbrace{\frac{1}{N}\sum_{n=1}^N \alpha_n \left(1 - y_n x_n^\top w\right) + \ldots}_{G(w,\alpha)}$$

The function G is convex in w and concave in α

Q2: Can the min and max be interchanged?

Always true:

$$\max_{\alpha} \min_{w} G(w,\alpha) \leq \min_{w} \max_{\alpha} G(w,\alpha)$$

Equality if G is convex in w, concave in α and the domains of w and α are convex and compact: $\max \min G(w,\alpha) = \min \max G(w,\alpha)$

Q2: Can the min and max be interchanged?

Always true:
$\max \min G(w,\alpha) \leq \min \max G(w,\alpha)$
$\alpha \quad w$
w $\quad \alpha$
Proof:
$\min G(\alpha, w) \leq G(\alpha, w')$ for any w'
w
$\max \min G(\alpha, w) \leq \max G(\alpha, w')$ for any w'
αw
$\max \min G(\alpha, w) \leq \min \max G(\alpha, w')$
$\alpha \quad w$
$w' \quad \alpha$

Application to SVM

For SVM, the condition is met, allowing us to interchange min and max:

$$\min_{w} L(w) = \max_{\alpha \in [0,1]^n} \min_{w} \frac{1}{N} \sum_{n=1}^{N} \alpha_n \left(1 - y_n x_n^\top w\right) + \frac{\lambda}{2} \|w\|_2^2$$

Minimizer computation:

$$\mathbf{Y} = \operatorname{diag}(\mathbf{y})$$

$\nabla_w G(w,\alpha) = -\frac{1}{N} \sum_{n=1}^{N} \alpha_n y_n x_n + \lambda w = 0 \implies w(\alpha) = \frac{1}{\lambda N} \sum_{n=1}^{N} \alpha_n y_n x_n = \frac{1}{\lambda N} \mathbf{X}^\top \mathbf{Y} \alpha$
Dual optimization problem:

$$\begin{aligned}\min_{w} L(w) &= \max_{\alpha \in [0,1]^n} \frac{1}{N} \sum_{n=1}^{N} \alpha_n \left(1 - \frac{1}{\lambda N} y_n x_n^\top \mathbf{X}^\top \mathbf{Y} \alpha\right) + \frac{1}{2\lambda N^2} \left\|\mathbf{X}^\top \mathbf{Y} \alpha\right\|_2^2 \\ &= \max_{\alpha \in [0,1]^n} \frac{\mathbf{1}^\top \alpha}{N} - \frac{1}{\lambda N^2} \alpha^\top \mathbf{YXX}^\top \mathbf{Y} \alpha + \frac{1}{2\lambda N^2} \left\|\mathbf{X}^\top \mathbf{Y} \alpha\right\|_2^2 \\ &= \max_{\alpha \in [0,1]^n} \frac{\mathbf{1}^\top \alpha}{N} - \frac{1}{2\lambda N^2} \alpha^\top \underbrace{\mathbf{YXX}^\top \mathbf{Y}}_{\text{PSD matrix}} \alpha\end{aligned}$$

Q3: Why?

$$\max_{\alpha \in [0,1]^n} \alpha^\top \mathbf{1} - \frac{1}{2\lambda N} \alpha^\top \underbrace{\mathbf{YXX}^\top \mathbf{Y}}_{\text{PSD matrix}} \alpha$$

1. Differentiable Concave Problem: Efficient solutions can be achieved using

- Quadratic programming solvers
- Coordinate ascent

2. Kernel Matrix Dependency: The cost function only depends on the data via the kernel matrix $K = \mathbf{XX}^\top \in \mathbb{R}^{N \times N}$ - no dependency on d
3. Dual Formulation Insight: α is typically sparse and non-zero exclusively for the training examples that are crucial in determining the decision boundary

Interpretation of the dual formulation

For any (x_n, y_n), there is a corresponding α_n given by

$$\max_{\alpha_n \in [0,1]} \alpha_n \left(1 - y_n x_n^\top w\right)$$

- If x_n is on the correct side and outside the margin, $1 - y_n x_n^\top w < 0$, then $\alpha_n = 0$
- If x_n is on the correct side and on the margin, $1 - y_n x_n^\top w = 0$, then $\alpha_n \in [0,1]$
- If x_n is strictly inside the margin or or the incorrect side, $1 - y_n x_n^\top w > 0$, then $\alpha_n = 1$

$\rightarrow$ The points for which $\alpha_n > 0$ are referred to as support vectors

$$(\alpha_n = 0 \text{ and } y_n = -1) \text{ or } (\alpha_n = 1 \text{ and } y_n = 1)$$

$w^\top x = -1 \quad w^\top x = 0$

The SVM hyperplane is supported by the support vectors

$$(\alpha_n = 0 \text{ and } y_n = 1) \text{ or } (\alpha_n = 1 \text{ and } y_n = -1)$$

$$w = \frac{1}{\lambda N} \sum_{n=1}^{N} \alpha_n y_n x_n$$

$\Rightarrow$ w does not depend on the observation (x_n, y_n) if $\alpha_n = 0$

$$(\alpha_n = 0 \text{ and } y_n = -1) \text{ or } (\alpha_n = 1 \text{ and } y_n = 1)$$

$$w^\top x = -1 \quad w^\top x = 0$$

Recap

- Hard SVM - finds max-margin separating hyperplane $\min_w \frac{1}{2}\|w\|^2$ such that $\forall n, y_n x_n^\top w \geq 1$
- Soft SVM - relax the constraint for non-separable data

$$\min_{w} \frac{\lambda}{2} \|w\|^2 + \frac{1}{N} \sum_{n=1}^{N} \left[1 - y_n x_n^\top w\right]_+$$

- Hinge loss can be optimized with (stochastic) sub-gradient method
- Duality: min max problem is equivalent to max min (convex-concave objective)
- Efficient solutions with quadratic programming and coordinate ascent
- The cost depends on the data via the kernel matrix (no dependency on d)

Supervised machine learning

We observe some data $S_{\text{train}} = \{x_n, y_n\}_{n=1}^N \in \mathscr{X} \times \mathscr{Y}$
Goal: given a new observation x, we want to predict its label y
How:

Nearest neighbor function

$$\begin{aligned}\text{nbh}_{S_{\text{train}}, k} &: X \rightarrow X^k \\ &x \mapsto \{\text{ the } k \text{ elements of } S_{\text{train}} \text{ closest to } x\}\end{aligned}$$

Nearest neighbor function

$$\begin{aligned}\text{nbh}_{S_{\text{train}}, k} &: X \rightarrow \mathscr{X}^k \\ &x \mapsto \{\text{ the } k \text{ elements of } S_{\text{train}} \text{ closest to } x\}\end{aligned}$$

$\text{nbh}_{S_{\text{train}}, 3}(x) = ?$

Nearest neighbor function

$$\begin{aligned}\text{nbh}_{S_{\text{train}}, k} &: X \rightarrow \mathscr{X}^k \\ &x \mapsto \{\text{ the } k \text{ elements of } S_{\text{train}} \text{ closest to } x\}\end{aligned}$$

$\text{nbh}_{S_{\text{train}}, 3}(x) = \{x_3, x_4, x_7\}$

Nearest neighbor function

$$\begin{aligned}\text{nbh}_{S_{\text{train}}, k} &: X \rightarrow \mathscr{X}^k \\ &x \mapsto \{\text{ the } k \text{ elements of } S_{\text{train}} \text{ closest to } x\}\end{aligned}$$

$\text{nbh}_{S_{\text{train}}, 2}(x) = ?$

Nearest neighbor function

$$\begin{aligned}\text{nbh}_{S_{\text{train}}, k} &: X \rightarrow \mathscr{X}^k \\ &x \mapsto \{\text{ the } k \text{ elements of } S_{\text{train}} \text{ closest}\end{aligned}$$

$\text{nbh}_{S_{\text{train}}, 2}(x) = ?$

Nearest neighbor function

$$\text{nbh}_{S_{\text{train}}, k} : \mathscr{X} \rightarrow \mathscr{X}^k$$

$x \mapsto \{$ the k elements of S_{train} closest to $x\}$

$$\text{nbh}_{S_{\text{train}}, 2}(x) = \{x_5, x_8\}$$

Nearest neighbor function

$$\begin{aligned}\text{nbh}_{S_{\text{train}}, k} &: X \rightarrow \mathscr{X}^k \\ &x \mapsto \{\text{ the } k \text{ elements of } S_{\text{train}} \text{ closest}\end{aligned}$$

Remarks:

- Different metrics can be employed
- High computational complexity for large N (but efficient data structure may exist)

k-NN can be used for regression ($y \in \mathbb{R}$)

$$f_{S_{\text{train}}, k}(x) = \frac{1}{k} \sum_{n : x_n \in nbh_{S_{\text{train}}, k}(x)} y_n$$

k-NN can be used for regression ($y \in \mathbb{R}$)

$$f_{S_{\text{train}}, k}(x) = \frac{1}{k} \sum_{n : x_n \in nbh_{S_{\text{train}}, k}(x)} y_n$$

$$f_{S_{\text{train}}, 3}(x) = ?$$

k-NN can be used for regression ($y \in \mathbb{R}$)

$$f_{S_{\text{train}}, k}(x) = \frac{1}{k} \sum_{n : x_n \in nbh_{S_{\text{train}}, k}(x)} y_n$$

k-NN can be used for regression ($y \in \mathbb{R}$)

$$f_{S_{\text{train}}, k}(x) = \frac{1}{k} \sum_{n : x_n \in nbh_{S_{\text{train}}, k}(x)} y_n$$

k-NN can be used for classification ($y \in \{0,1\}$)

$$f_{S_{\text{train}}, k}(x) = \text{majority} \{y_n : x_n \in \text{nbh}_{S_{\text{train}}, k}(x)\}$$

k-NN can be used for classification ($y \in \{0,1\}$)

$$f_{S_{\text{train}},k}(x) = \text{majority}\left\{y_n : x_n \in \text{nbh}_{S_{\text{train}},}(x)\right\}$$

$$f_{S_{\text{train}},1}(x) = ?$$

k-NN can be used for classification ($y \in \{0,1\}$)

$$f_{S_{\text{train}},k}(x) = \text{majority}\left\{y_n : x_n \in \text{nbh}_{S_{\text{train}},}(x)\right\}$$

$$f_{S_{\text{train}}\,1}(x) = 1$$

k-NN can be used for classification ($y \in \{0,1\}$)

$$f_{S_{\text{train}},k}(x) = \text{majority}\left\{y_n : x_n \in \text{nbh}_{S_{\text{train}},}(x)\right\}$$

$$f_{S_{\text{train}}\,3}(x) = ?$$

k-NN can be used for classification ($y \in \{0,1\}$)

$$f_{S_{\text{train}},k}(x) = \text{majority}\left\{y_n : x_n \in \text{nbh}_{S_{\text{train}},}(x)\right\}$$

$$f_{S_{\text{train}}\,3}(x) = 0$$

k-NN can be used for classification ($y \in \{0,1\}$)

$$f_{S_{\text{train}},k}(x) = \text{majority}\left\{y_n : x_n \in \text{nbh}_{S_{\text{train}},}(x)\right\}$$

$$f_{S_{\text{train}}\,4}(x) = ?\ \text{Tie!}$$

k-NN can be used for classification ($y \in \{0,1\}$)

$$f_{S_{\text{train}},k}(x) = \text{ majority } \{y_n : x_n \in \text{nbh}_{S_{\text{train}},k}(x)\}$$

Remarks:

- Choose an odd value for k to prevent ties
- Generalization: smoothing kernels; weighted linear combination of elements

Why does it make sense?

- Relevant in the presence of spatial correlation
- Implicitly models intricate decision boundaries in low-dimensional spaces

Bias-variance tradeoff in k-NN

For small k:

- Low bias - complex decision boundary
- High variance - overfitting

For large k:
(When $k = N$, prediction is constant)

- High bias
- Low variance

1-nearest neighbor classification

U-shaped curve for $\mathrm{k-NN}$ bias-variance tradeoff

Complexity increases as k decreases

Find a k that balances bias and variance

Characteristics of an optimal k:

- Low bias: Ensures a sufficiently complex decision boundary
- Low variance: Prevents overfitting

Summary: k-Nearest Neighbor

Pros:

- No optimization or training
- Easy to implement
- Works well in low dimensions, allowing for very complex decision boundaries

Cons:

- Slow at query time
- Not suitable for high-dimensional data
- Choosing the right local distance is crucial

Curse of dimensionality

Claim 1: As the dimensionality grows, fixed-size training sets cover a diminishing fraction of the input space
Assume the data $x \sim \mathcal{U}\left([0,1]^d\right)$
Consider a blue box around the center x_0 of size r

$$\mathbb{P}(x \in) = r^d := \alpha$$

If $\alpha = 0.01$, to have:

$$d = 10,\ \text{we need } r = 0.63$$
$$d = 100,\ \text{we need } r = 0.95$$

We need to explore almost the whole box
$X = [0,1]^d$

Curse of dimensionality

Claim 2: In high-dimension, data-points are far from each other.
Consider N i.i.d. points uniform in the $[0,1]^d$

$$\mathbb{P}\left(\exists x_i \in \square\right) \geq 1/2 \Longrightarrow r \geq \left(1 - \frac{1}{2^{1/N}}\right)^{1/d}$$

Proof: $\mathbb{P}(x \notin \square) = 1 - r^d$

$$\mathbb{P}\left(x_i \notin \quad, \forall i \leq N\right) = \left(1 - r^d\right)^N$$
$$\mathbb{P}\left(\exists x_i \in \square\right) = 1 - \left(1 - r^d\right)^N$$

For $d = 10, N = 500$, we have $r \geq 0.52$

$$\mathscr{X} = [0,1]^d$$

Generalization bound for 1-NN

Setup: $(X,Y) \sim \mathscr{D}$ over $\mathscr{X} \times \mathscr{Y} = [0,1]^d \times \{0,1\}$
Goal: Bound the classification error:

$$L(f) = \mathbb{P}_{(X,Y)\sim D}(Y \neq f(X))$$

Baseline:

- Bayes classifier: minimizes L over all classifiers

$f_*(x) = 1_{\eta(x)\geq 1/2}$ where $\eta(x) = \mathbb{P}(Y = 1 \mid X = x)$

- Bayes risk: represents the minimum probability of misclassification

$L\left(f_*\right) = \mathbb{P}\left(f_*(X) \neq Y\right) = \mathbb{E}_{X\sim D_X}[\min\{\eta(X), 1-\eta(X$ We want to bound

Generalization bound for 1-NN

- Bayes classifier: minimizes L over all classifiers

$f_*(x) = 1_{\eta(x)\geq 1/2}$ where $\eta(x) = \mathbb{P}(Y = 1 \mid X = x)$

- Bayes risk: represents the minimum probability of misclassification

$L\left(f_*\right) = \mathbb{P}\left(f_*(X) \neq Y\right) = \mathbb{E}_{X\sim \mathscr{D}_X}[\min\{\eta(X), 1-\eta(X)\}]$

Generalization bound for 1-NN

Proof 2:

$$L\left(f_*\right) = \mathbb{E}_{(X,Y)\sim\mathscr{D}}\left[1_{f_*(X)\neq Y}\right]$$

$$= \mathbb{E}_{X\sim\mathscr{D}_X}\left[\mathbb{E}_{Y\sim\mathscr{D}_{Y|X}}\left[1_{f_*(X)\neq Y} \mid X\right] 1_{\eta(X)\geq 1/2} + E\right.$$
$$= \mathbb{E}_{X\sim\mathscr{D}_X}\left[\mathbb{E}_{Y\sim\mathscr{D}_{Y|X}}\left[1_{1\neq Y} \mid X\right] 1_{\eta(X)\geq 1/2} + E_{Y\sim\mathscr{D}}\right.$$
$$= \mathbb{E}_{X\sim D_X}\left[\mathbb{P}(Y = 0 \mid X)1_{\eta(X)\geq 1/2} + \mathbb{P}(Y = 1 \mid X)1\right.$$
$$= \mathbb{E}_{X\sim\mathscr{D}_X}[\min\{\eta(X), 1-\eta(X)\}]$$

- Bayes risk: represents the minimum probability of misclassification

$L\left(f_*\right) = \mathbb{P}\left(f_*(X) \neq Y\right) = \mathbb{E}_{X\sim D_X}[\min\{\eta(X), 1-\eta($

Generalization bound for 1-NN

Assumption: $\exists c \geq 0, \forall x, x' \in \mathscr{X}$:

$$\left|\eta(x) - \eta\left(x'\right)\right| \leq c\left\|x - x'\right\|_2$$

$\Rightarrow$ Nearby points are likely to share the same label
Claim:

$$\leq 2L\left(f_*\right) + 4c\sqrt{d}N^{-\frac{1}{d+1}}$$

Interpretation:
For constant d and $N \to \infty$: $\mathbb{E}_{S_{\text{train}}}\left[L\left(f_{S_{\text{train}}}\right)\right] \leq 2L\left(f_*\right)$
geometric term: average distance between a random point and its closest neighbor
To achieve a constant error, we need $N \propto d^{(d+1)/2}$ - curse of dimensionality
Despite common belief: Interpolation method can generalize well

Proof

$$\mathbb{E}_{S_{\text{train}}}\left[L\left(f_{S_{\text{train}}}\right)\right] = \mathbb{E}_{S_{\text{train}}}\left[\mathbb{P}_{(X,Y)\sim\mathscr{D}}\left[f_{S_{\text{train}}}(X)\right.\right.$$

We first sample N unlabeled examples $S_{\text{train},X} = (X_1, \cdots X_N) \sim \mathscr{D}_X$, an unlabeled example $X \sim \mathscr{D}_X$ and define $X' = \text{nbh}_{S_{\text{train}},1}(X)$
Finally we sample $Y \sim \eta(X)$ and $Y' \sim \eta\left(X'\right)$
We have:

$$\mathbb{E}_{S_{\text{train}}}\left[L\left(f_{S_{\text{train}}}\right)\right] = \mathbb{E}_{S_{\text{train}},X, X\sim D_X, Y\sim\eta(X), Y'\sim\eta(}$$
$$= \mathbb{E}_{S_{\text{train},X}, X\sim\mathscr{D}_X, Y\sim\eta(X), Y'\sim\eta(X')}\left[1_{Y\neq Y'}\right]$$
$$= \mathbb{E}_{S_{\text{train}},X, X\sim\mathscr{D}_X}\left[\mathbb{P}_{Y\sim\eta(X), Y'\sim\eta(X')}\left(Y \neq Y'\right)\right]$$

Proof

Consider two points $x, x' \in [0,1]^d$.
Sample their labels $Y \sim \eta(x)$ and $Y' \sim \eta(x')$
Claim:

$$\mathbb{P}\left(Y' \neq Y\right) \leq 2\min\{\eta(x), 1-\eta(x)\} + c\left\|x - x'\right\|$$

- Simple case: $x = x'$

$$\begin{aligned}\mathbb{P}\left(Y' \neq Y\right) &= \mathbb{E}\left[1_{Y' \neq Y} 1_{Y'=1} + 1_{Y' \neq Y'} 1_{Y'=0}\right] \\ &= \mathbb{P}\left(Y'=1\right)\mathbb{P}(Y=0) + \mathbb{P}\left(Y'=1\right)\mathbb{P}(Y=0) \\ &= 2\eta(x)(1-\eta(x)) \\ &\leq 2\min\{\eta(x), 1-\eta(x)\}\end{aligned}$$

Case 1:
$\mathbf{Y} = \mathbf{0}(1-\eta(x))$
$Y' = 1 \quad \eta(x)$
Case 2:
$\mathrm{Y} = \mathbf{1} \quad \eta(x)$
$Y' = 0 \quad (1-\eta(x))$

Proof

- General case:

$$\begin{aligned}\mathbb{P}\left(Y \neq Y'\right) =& \eta(x)\left(1-\eta\left(x'\right)\right) + \eta\left(x'\right)(1-\eta(x)) \\ =& \eta(x)(1-\eta(x)) + \eta(x)\left(\eta(x) - \eta\left(x'\right)\right) \\ &+ \eta(x)(1-\eta(x)) + \left(\eta\left(x'\right) - \eta(x)\right)(1-\eta(x)) \\ =& 2\eta(x)(1-\eta(x)) + (2\eta(x)-1)\left(\eta(x) - \eta\left(x'\right)\right) \\ &\leq 2\eta(x)(1-\eta(x)) + |(2\eta(x)-1)|\left|\eta(x) - \eta\left(x'\right)\right| \\ &\leq 2\eta(x)(1-\eta(x)) + \left|\eta(x) - \eta\left(x'\right)\right| \\ &\leq 2\eta(x)(1-\eta(x)) + c\left\|x - x'\right\| \\ &\leq 2\min\{\eta(x), 1-\eta(x)\} + c\left\|x - x'\right\|\end{aligned}$$

Proof

$$\begin{aligned}\mathbb{E}_{S_{\text{train}}}\left[L\left(f_{S_{\text{train}}}\right)\right] &= \mathbb{E}_{S_{\text{train}},\, ,X, X \sim \mathscr{D}_X, Y \sim \eta(X), Y' \sim \eta(X')}\left[1_{Y \neq f_{\text{train}}(X)}\right] \\ &= \mathbb{E}_{S_{\text{train}},\, ,X, X \sim \mathscr{D}_X, Y \sim \eta(X), Y' \sim \eta(X')}\left[1_{Y \neq Y'}\right] \\ &= \mathbb{E}_{S_{\text{train}}\, ,X, X \sim D_X}\left[\mathbb{P}_{Y \sim \eta(X), Y' \sim \eta(X')}(Y \neq Y')\right] \\ &\leq \mathbb{E}_{S_{\text{train}}\, ,X, X \sim \mathscr{D}_X}\left[2\min\{\eta(X), 1-\eta(X)\} + c\left\|X - X'\right\|\right] \\ &\leq 2L\left(f_*\right) + c\mathbb{E}_{S_{\text{train}}\, ,X \sim \mathscr{D}_X}\left[\left\|X - \text{nbh}_{S_{\text{train}}}(X)\right\|\right]\end{aligned}$$

Bound on the geometric term

Consider a fresh sample $X \sim \mathscr{D}$ and denote by $p_k = \mathbb{P}\left(X \in \text{Box }_k\right)$
Consider the box which contains X. Two options:

- The box contains an element of S_{train} X has a neighbor in S_{train} at distance at most $\sqrt{d}\varepsilon$

It happens with probability $1-\left(1-p_k\right)^N$
Proof: Consider the worst case:

$$\left\|X - x_i\right\| = \sqrt{\sum_{i=1}^{d} \varepsilon^2} = \sqrt{d}\varepsilon$$

Bound on the geometric term

Consider a fresh sample $X \sim \mathscr{D}$ and denote by $p_k = \mathbb{P}\left(X \in \text{Box }_k\right)$
Consider the box which contains X. Two options:

- The box contains an element of S_{train} X has a neighbor in S_{train} at distance at most $\sqrt{d}\varepsilon$

It happens with probability $1-\left(1-p_k\right)^N$

- There is no element of S_{train}. The nearest neighbor of X can be at worst at a distance $\sqrt{d}$ It happens with probability $\left(1-p_k\right)^N$

Bound on the geometric term

Consider a fresh sample $X \sim \mathscr{D}$ and denote by

$$p_k = \mathbb{P}\left(X \in \text{Box}_k\right)$$

Bound on the geometric term

$\mathbb{E}[\|X - \text{nbh}(X)\|] \leq \sum_k p_k\left[\left(1-p_k\right)^N \sqrt{d} + \left(1-\left(1-p_k\right)^N\right)\sqrt{d}\varepsilon\right]$
Claim: The bound is derived by optimizing over p_k and ε
Intuition:

- If p_k is large: it is likely that we pick that box but it is also likely that we find a training point in that box
- If p_k is small, we are generally safe, as by its definition, this scenario occurs infrequently

Nearest Neighbors is a local averaging method

Local averaging methods aim to approximate the Bayes predictor directly - without the need for optimization.
This is achieved by approximating the conditional distribution $p(y \mid x)$ by some $\hat{p}(y \mid x)$ These "plug-in" estimators are:
. $f(x) \in \arg\max \hat{\mathbb{P}}(Y = y \mid x)$ for classification with the $0-1$ loss $y \in \mathcal{Y}$

- $f(x) = \hat{\mathbb{E}}[Y \mid x] = \int y\hat{p}(y \mid x)dy$ for regression with the square loss

In the case of nearest neighbors:

$$\hat{p}(y \mid x) = \sum_{n=1}^{N} \hat{w}_n(x) 1_{y = y_n}$$

where $\hat{w}(x) = 1/k$ for the k nearest neighbors (0 otherwise)

Recap

- k-NN: a local averaging method for regression and classification
- use a notion of distance to define neighborhoods (k nearest neighbors)
- the prediction is a function of these neighborhoods

e.g., majority selection for classification, weighted sum for regression

- Bias-variance: small/large k leads to low/high bias and high/low variance
- Curse of dimensionality: as $d \nearrow \infty$, it is harder to define local neighborhoods
- For $N \rightarrow \infty, 1-\mathrm{NN}$ is competitive with Bayes classifier
- N needs to scale exponentially in d to achieve the same error

---

Equivalent formulations for ridge regression

Objective

$$\min_w \frac{1}{2N} \sum_{n=1}^{N}\left(y_n - w^\top x_n\right)^2 + \frac{\lambda}{2}\|w\|^2$$

The solution is given by

$$\mathcal{W}_* = \frac{1}{N}\left(\frac{1}{N}\mathbf{X}^\top \mathbf{X} + \lambda I_d\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

$\mathbf{X}^\top \in \mathbb{R}^{d \times N} \rightarrow d \times d$

Alternatively, the solution can be written as

$$\mathcal{W}_* = \frac{1}{N}\mathbf{X}^\top (\underbrace{\frac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda I_N}_{\mathbf{X} \in \mathbb{R}^{N \times d}})^{-1}\mathbf{y}$$

Proof: Let $P \in \mathbb{R}^{m \times n}$ and $Q \in \mathbb{R}^{n \times m}$

$$P\left(QP + I_n\right) = PQP + P = \left(PQ + I_m\right)P$$

Assuming that both $QP + I_n$ and $PQ + I_m$ are invertible

$$\left(PQ + I_m\right)^{-1} P = P\left(QP + I_n\right)^{-1}$$

We deduce the result with $P = \mathbf{X}^\top$ and $Q = \frac{1}{\lambda N}\mathbf{X}$

$$\mathcal{W}_* = \frac{1}{N}\underbrace{(\frac{1}{N}\mathbf{X}^\top \mathbf{X} + \lambda I_d)^{-1} \mathbf{X}^\top \mathbf{y}}_{\rightarrow d \times d}$$

$\mathbf{X}^\top \in \mathbb{R}^{d \times N} \rightarrow d^{-1}$

But it can be alternatively written as

$$\mathcal{W}_* = \frac{1}{N}\mathbf{X}^\top (\underbrace{\frac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda I_N}_{\mathbf{X} \in \mathbb{R}^{N \times d}})^{-1}\mathbf{y}$$

Usefulness of the alternative form

$$\mathcal{W}_* = \underbrace{\frac{1}{N}\mathbf{X}^\top}_{d \times N} (\underbrace{\frac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda I_N}_{N \times N})^{-1}\mathbf{y}$$

1. Computational complexity:

- For the original formulation $\frac{1}{N}\left(\frac{1}{N}\mathbf{X}^\top \mathbf{X} + \lambda I_d\right)^{-1} \mathbf{X}^\top \mathbf{y}, O\left(d^3 + Nd^2\right)$
- For the new formulation $\frac{1}{N}\mathbf{X}^\top\left(\frac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda I_N\right)^{-1}\mathbf{y}, O\left(N^3 + dN^2\right)$

$\Rightarrow$ Depending on d, N one formulation may be more efficient than the other

2. Structural difference:

$$w_* = \mathbf{X}^\top \alpha_* \text{ where } \alpha_* = \frac{1}{N}\left(\frac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda I_N\right)^{-1}\mathbf{y}$$

$\Rightarrow w_* \in \operatorname{span}\left\{x_1, \cdots, x_N\right\}$
These two insights are fundamental to understanding the kernel trick

Representer Theorem

Claim: For any loss function ℓ, there exists $\alpha_* \in \mathbb{R}^N$ such that

$$w_* := \mathbf{X}^\top \alpha_* \in \arg\min_w \frac{1}{N}\sum_{n=1}^{N} \ell\left(x_n^\top w, y_n\right) + \frac{\lambda}{2}\|w\|^2$$

Meaning: There exists an optimal solution within $\operatorname{span}\left\{x_1, \cdots, x_N\right\}$
Consequence: This is more general than LS, enabling the kernel trick to various problems, including Kernel SVM, Kernel LS, and Kernel Principal Component Analysis

Proof of the representer theorem

Let w_* be an optimal solution of $\min_w \frac{1}{N}\sum_{n=1}^{N} \ell\left(x_n^\top w, y_n\right) + \frac{\lambda}{2}\|w\|^2$

We can always rewrite w_* as $w_* = \sum_{n=1}^{N} \alpha_n x_n + u$ where $u^\top x_n = 0$ for all n

Let's define $w = w_* - u$

- $\|w_*\|^2 = \|w\|^2 + \|u\|^2$, thus $\|w\|^2 \leq \|w_*\|^2$
- For all $n, w^\top x_n = (w_* - u)^\top x_n = w_*^\top x_n$, thus $\ell\left(x_n^\top w, y_n\right) = \ell\left(x_n^\top w_*, y_n\right)$

Therefore

$$\frac{1}{N}\sum_{n=1}^{N} \ell\left(x_n^\top w, y_n\right) + \frac{\lambda}{2}\|w\|^2 \leq \frac{1}{N}\sum_{n=1}^{N} \ell\left(x_n^\top w_*, y_n\right) + \frac{\lambda}{2}\|w_*\|^2$$

And w is an optimal solution for this problem.

Kernelized ridge regression

Classic formulation in w :

$$w_* = \arg\min_w \frac{1}{2N}\|\mathbf{y} - \mathbf{X}w\|^2 + \frac{\lambda}{2}\|w\|^2$$

Alternative formulation in α :

$$\alpha_* = \arg\min_\alpha \frac{1}{2}\alpha^\top \left(\frac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda I_N\right)\alpha - \frac{1}{N}\alpha^\top \mathbf{y}$$

Claim: These two formulations are equivalent

Proof: Set the gradient to 0 , to obtain $\alpha_* = \frac{1}{N}\left(\frac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda I_N\right)^{-1}\mathbf{y}$, and $w_* = \mathbf{X}^\top \alpha_*$

Key takeaways:

- Computational complexity - depending on d, N
- The dual formulation only uses $\mathbf{X}$ through the kernel matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$

Kernel matrix

$$\mathbf{K} = \mathbf{X}\mathbf{X}^\top = \begin{pmatrix} x_1^\top x_1 & x_1^\top x_2 & \cdots & x_1^\top x_N \\ x_2^\top x_1 & x_2^\top x_2 & \cdots & x_2^\top x_N \\ \vdots & \vdots & \ddots & \vdots \\ x_N^\top x_1 & x_N^\top x_2 & \cdots & x_N^\top x_N \end{pmatrix} = \left(x_i^\top x_j\right)_{i,j} \in \mathbb{R}^{N\times N}$$

Embedding into feature spaces

Usefulness of feature spaces

Kernel matrix with feature spaces

When a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$ is used,

$$(x_n)_{n=1}^N \hookrightarrow (\phi(x_n))_{n=1}^N$$

The associated kernel matrix is

$$\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top = \begin{pmatrix} \phi(x_1)^\top \phi(x_1) & \phi(x_1)^\top \phi(x_2) & \cdots & \phi(x_1)^\top \phi(x_N) \\ \phi(x_2)^\top \phi(x_1) & \phi(x_2)^\top \phi(x_2) & \cdots & \phi(x_2)^\top \phi(x_N) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(x_N)^\top \phi(x_1) & \phi(x_N)^\top \phi(x_2) & \cdots & \phi(x_N)^\top \phi(x_N) \end{pmatrix} \in \mathbb{R}^{N\times N}$$

Problem: when $d \lll \tilde{d}$ computing $\phi(x)^\top \phi(x')$ costs $O(\tilde{d})$ - too expensive

Kernel trick

Kernel function: $\kappa(x, x')$ such that

$\kappa(x, x') = \phi(x)^\top \phi(x')$

It is equivalent to

- Directly compute $\kappa(x, x')$
- First map the features to $\phi(x)$, then compute $\phi(x)^\top \phi(x')$

Purpose: enable computation of linear classifiers in high-dimensional space without performing computations in this high-dimensional space directly.

Predicting with kernels

Problem: The prediction is $y = \phi(x)^\top w_*$ but computing $\phi(x)$ can be expensive

Question: How can we make predictions using only the kernel function, without the need to compute $\phi(x)$?

Answer: $\phi(x)^\top w_* = \phi(x)^\top \phi(\mathbf{X})^\top \alpha_* = \sum_{n=1}^{N} \kappa(x, x_n)\, \alpha_{*n}$ We can do a prediction only using the kernel function

Important remark:

$$\underline{y = \phi(x)^\top w_* = f_{W_*}(x)}$$

Linear prediction

in the feature space Non linear prediction

in the $\mathscr{X}$ space

Examples of kernel (easy)

1. Linear kernel: $\kappa(x, x') = x^\top x'$

$\rightarrow$ Feature map is $\phi(x) = x$

2. Quadratic kernel: $\kappa(x, x') = (xx')^2$ for $x, x' \in \mathbb{R}$

$\Rightarrow$ Feature map is $\phi(x) = x^2$

3. Polynomial kernel

Let $x, x' \in \mathbb{R}^3$

$$\kappa(x, x') = \left(x_1 x_1' + x_2 x_2' + x_3 x_3'\right)^2$$

Feature map:

Proof:

$$\phi(x) = \left[x_1^2, x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3\right] \in \mathbb{R}^6$$

$$\begin{aligned} \kappa(x, x') &= \left(x_1 x_1' + x_2 x_2' + x_3 x_3'\right)^2 \\ &= \left(x_1 x_1'\right)^2 + \left(x_2 x_2'\right)^2 + \left(x_3 x_3'\right)^2 + 2x_1x_2x_1'x_2' + 2x_1x_3x_1'x_3' + 2x_2x_3x_2'x_3' \\ &= \left(x_1^2, x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3\right)^\top \left(x_1'^2, x_2'^2, x_3'^2, \sqrt{2}x_1'x_2', \sqrt{2}x_1'x_3', \sqrt{2}x_2'x_3'\right) \end{aligned}$$

We obtain ϕ by identification

4. Radial basis function (RBF) kernel

Let $x, x' \in \mathbb{R}^d$

$$\kappa(x, x') = e^{-(x-x')^\top (x-x')}$$

For $x, x' \in \mathbb{R}$

$$\kappa(x, x') = e^{-(x-x')^2}$$

Feature map:

$$\phi(x) = e^{-x^2}\left(\cdots, \frac{2^{k/2}x^k}{\sqrt{k!}} \cdots\right)$$

Proof: $\quad \kappa(x, x') = e^{-x^2 - x'^2 + 2xx'}$

$$= e^{-x^2} e^{-x'^2} \sum_{k=0}^{\infty} \frac{2^k x^k x'^k}{k!} \text{ by the Taylor expansion of } \exp$$

$$\phi(x) = e^{-x^2}\left(\cdots, \frac{2^{k/2}x^k}{\sqrt{k!}} \cdots\right) \Longrightarrow \phi(x)^\top \phi(x') = \kappa(x, x')$$

Interest: it cannot be represented as an inner product in a finite-dimensional space

Building new kernels from existing kernels

Let κ_1, κ_2 be two kernel functions and ϕ_1, ϕ_2 the corresponding feature maps

Claim 1: Positive linear combinations of kernel are kernels

$$\kappa(x, x') = \alpha\kappa_1(x, x') + \beta\kappa_2(x, x') \text{ for } \alpha, \beta \geq 0$$

Claim 2: Products of kernels are kernels

$$\kappa(x, x') = \kappa_1(x, x')\,\kappa_2(x, x')$$

Objective: To provide building blocks for deriving new kernels

Proof 1:

$$\begin{aligned} \kappa(x, x') &= \alpha\kappa_1(x, x') + \beta\kappa_2(x, x') \\ &= \alpha\phi_1(x)^\top \phi_1(x') + \beta\phi_2(x)^\top \phi_2(x') \\ &= \phi(x)^\top \phi(x') \end{aligned}$$

where $\phi(x) = \begin{pmatrix} \sqrt{\alpha}\phi_1(x) \\ \sqrt{\beta}\phi_2(x) \end{pmatrix} \in \mathbb{R}^{d_1+d_2}$

kernels from old kernel

s and ϕ_1, ϕ_2 the corresponding feature maps

Claim 1: Positive linear combinations of kernel are kernel

$$\kappa(x, x') = \alpha\kappa_1(x, x') + \beta\kappa_2(x, x')$$

Claim 2: Products of kernels are kernel

$$\kappa(x, x') = \kappa_1(x, x')\,\kappa_2(x, x')$$

Proof 2:

$\kappa(x, x') = \kappa_1(x, x')\,\kappa_2(x, x')$

$$= \phi_1(x)^\top \phi_1(x')\,\phi_2(x)^\top \phi_2(x')$$

Let

$\phi(x)^\top = \left((\phi_1(x))_1 (\phi_2(x))_1, \cdots, (\phi_1(x))_1 (\phi_2(x))_{d_2}\right.$

$\mathbb{R}^{d_1 d_2}$ then

$$\begin{aligned} \phi(x)^\top \phi(x') &= \sum_{i,j} (\phi_1(x))_i (\phi_2(x))_j (\phi_1(x'))_i (\phi_2 (x \\ &= \sum_i (\phi_1(x))_i (\phi_1(x'))_i \sum_j (\phi_2(x))_j (\phi \\ &= \phi_1(x)^\top \phi_1(x')\,\phi_2(x)^\top \phi_2(x') = \kappa(x, \end{aligned}$$

Claim 2: Products of kernels are kernel

$$\kappa(x, x') = \kappa_1(x, x')\,\kappa_2(x, x')$$

Mercer's condition

Question: Given a kernel function κ, how can we ensure the existence of a feature map ϕ such that

$$\kappa(x, x') = \phi(x)^\top \phi(x')$$

Answer: It is true if and only if the following Mercer's conditions are fulfilled:

- The kernel function is symmetric:

$$\forall x, x', \kappa(x, x') = \kappa(x', x)$$

- The kernel matrix is psd for all possible input sets:

$$\forall N \geq 0, \forall (x_n)_{n=1}^N, \quad \mathbf{K} = (\kappa(x_i, x_j))_{i,j=1}^N \succcurlyeq 0$$

Recap

- Many algorithms (SVM, Least Squares, PCA, etc) can be rewritten so that they rely only on inner products between data points $(\mathbf{X}\mathbf{X}^\top)$
- This motivates to generalize the inner products with kernels to make the model nonlinear in the input space
- This can improve efficiency by avoiding a direct computation of feature maps $\phi(x)$ of a potentially high-dimensional space
- To predict with kernels, you need to compute the similarity $k(x, x_i)$ of a new point x with every training point x_i
- You can derive new kernels using a certain set of properties

Bonus: proof of Mercer theorem

- If κ represents an inner product then it is symmetric and the kernel matrix is psd:

$$v^\top \mathrm{K} v = \sum_{i,j} v_i v_j \phi(x_i)^\top \phi(x_j) = \left\| \sum_i v_i \phi(x_i) \right\|^2$$

- Define $\phi(x) = \kappa(\cdot, x)$. Define a vector space of functions by considering all linear combinations $\left\{\sum_i \alpha_i \kappa(\cdot, x_i)\right\}$. Define an inner product on this vector space by

$$\left\langle \sum_i \alpha_i \kappa(\cdot, x_i), \sum_j \beta_j \kappa(\cdot, x_j') \right\rangle = \sum_{i,j} \alpha_i \beta_j \kappa(x_i, x_j')$$

This is a valid inner product (symmetric, bilinear and positive definite, with equality holding only if $\phi(x)$ is the zero function)

Consequently

$$\langle \phi(x), \phi(x') \rangle = \langle \kappa(\cdot, x), \kappa(\cdot, x') \rangle = \kappa(x, x')$$

Recap

Neural Networks: Key Facts

Supervised learning : we observe some data $S_{\text{train}} = \{x_n, y_n\}_{n=1}^N \in \mathscr{X} \times \mathscr{Y} \Rightarrow$ given a new x, we want to predict its label y

Linear prediction (with augmented features): $y = f_{\text{Lin}}(x) = \phi(x)^\top w$ Features are given

Prediction with a NN:

$$y = f_{\text{NN}}(x) = f(x)^\top w$$

Fully Connected Neural Networks

Input

$$x_j^{(l)} = \phi\left(\sum_{i=1}^K x_i^{(l-1)} w_{i,j}^{(l)} + b_j^{(l)}\right)$$

Important: ϕ is non-linear otherwise we can only represent linear functions

weight of the edge going from node i in layer $l-1$ to node j in layer l

bias term associated with node j in layer l

NNs: Inference vs. Training

Linear prediction on features $f(x)$

$$h(x) = f(x)^\top w^{(L+1)} + b^{(L+1)}$$

Inference

$h(x)$

$\text{sign}(h(x))$

Binary Classification

with $y \in \{-1, 1\}$

Multi-Class Classification

with $y \in \{1, \ldots, K\}$

with $y \in \mathbb{R}$ Training

$\ell(y, h(x)) = (h(x) - y)^2$

$\text{argmax}_{c \in \{1,\ldots,K\}} h(x)_c$ $\ell(y, h(x)) = \log(1 + \exp(-yh(x)))$

$\ell(y, h(x)) = -\log \frac{e^{h(x)_y}}{\sum_{i=1}^K e^{h(x)_i}}$

With a suitable representation of the data $f(x)$ learned by the network, the last layer only performs a linear regression or classification step

Today: How do we train a NN?

Training of NNs

Training loss for a regression problem with $S_{\text{train}} = \{(x_n, y_n)\}_{n=1}^N$:

$$\mathscr{L}(f) = \frac{1}{2N} \sum_{n=1}^N (y_n - f(x_n))^2$$

where f is the function represented by a NN with weights $\left(w_{i,j}^{(l)}\right)$ and biases $\left(b_i^{(l)}\right)$

Task:

Remarks:

$$\min_{w_{i,j}^{(l)}, b_i^{(l)}} \mathscr{L}(f)$$

- Regularization can be added to avoid overfitting and is easy to implement
- Non-convex optimization problem

$\Rightarrow$ not guaranteed to converge to a global minimum

Training of NNs with SGD

SGD algorithm: Uniformly sample n, compute the gradient of $\mathscr{L}_n = \frac{1}{2}(y_n - f(x_n))^2$ to update:

$$\left(w_{i,j}^{(l)}\right)_{t+1} = \left(w_{i,j}^{(l)}\right)_t - \gamma \frac{\partial \mathscr{L}_n}{\partial w_{i,j}^{(l)}} \quad \left(b_i^{(l)}\right)_{t+1} = \left(b_i^{(l)}\right)_t - \gamma \frac{\partial \mathscr{L}_n}{\partial b_i^{(l)}}$$

In Practice: Step size schedule, mini-batch, momentum, Adam

Training of NNs with SGD

SGD algorithm: Uniformly sample n, compute the gradient of $\mathscr{L}_n = \frac{1}{2}(y_n - f(x_n))^2$ to update:

$$\left(w_{i,j}^{(l)}\right)_{t+1} = \left(w_{i,j}^{(l)}\right)_t - \gamma \frac{\partial \mathscr{L}_n}{\partial w_{i,j}^{(l)}} \quad \left(b_i^{(l)}\right)_{t+1} = \left(b_i^{(l)}\right)_t - \gamma \frac{\partial \mathscr{L}_n}{\partial b_i^{(l)}}$$

In Practice: Step size schedule, mini-batch, momentum, Adam

Problem: With $O\left(K^2 L\right)$ parameters, applying chain-rules independently is inefficient due to the compositional structure of f

Solution: the Backpropagation algorithm computes gradients via the chain rule but reuses intermediate computations

Description of NN parameters

Input

- $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times K}$

Weight matrices: $\mathbf{W}^{(l)}$ such that $\mathbf{W}_{i,j}^{(l)} = w_{i,j}^{(l)}$, of size

- $\mathbf{W}^{(l)} \in \mathbb{R}^{K \times K}$ for $2 \leq l \leq L$
- $\mathbf{W}^{(L+1)} \in \mathbb{R}^K$

Bias vectors: $b^{(l)}$ such that the i-th component is $b_i^{(l)}$

- $b^{(l)} \in \mathbb{R}^K$ for $1 \leq l \leq L$
- $b^{(L+1)} \in \mathbb{R}$

Compact description of output

The functions implemented by each layer can be written as:

- $x^{(1)} = f^{(1)}\left(x^{(0)}\right) := \phi\left(\left(\mathbf{W}^{(1)}\right)^\top x^{(0)} + b^{(1)}\right)$
- $x^{(l)} = f^{(l)}\left(x^{(l-1)}\right) := \phi\left(\left(\mathbf{W}^{(l)}\right)^\top x^{(l-1)} + b^{(l)}\right)$
- $y = f^{(L+1)}\left(x^{(L)}\right) := \left(\mathbf{W}^{(L+1)}\right)^\top x^{(L)} + b^{(L+1)}$

The overall function $y = f\left(x^{(0)}\right)$ is just the composition of the layer functions:

$$f = f^{(L+1)} \circ f^{(L)} \circ \cdots \circ f^{(l)} \circ \cdots \circ f^{(2)} \circ f^{(1)}$$

Cost function

Cost function:

$$\mathscr{L} = \frac{1}{2N} \sum_{n=1}^N \left(y_n - f^{(L+1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x_n)\right)^2$$

Remarks:

- The specific form of the loss is not crucial
- $\mathscr{L}$ is a function of all weight matrices and bias vectors
- Each function $f^{(l)}$ is parameterized by weights $\mathbf{W}^{(l)}$ and biases $b^{(l)}$

Individual loss for SGD:

$$\mathscr{L}_n = \frac{1}{2}\left(y_n - f^{(L+1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x_n)\right)^2$$

Goal: Compute for all (i, j, l)

$$\frac{\partial \mathscr{L}_n}{\partial w_{i,j}^{(l)}} \quad \text{and} \quad \frac{\partial \mathscr{L}_n}{\partial b_i^{(l)}}$$

Chain rule

$$\mathscr{L}_n = \frac{1}{2}\left(y_n - f^{(L+1)} \circ \cdots \circ f^{(l+1)} \circ \phi(\underbrace{\left(\mathbf{W}^{(l)}\right)^\top x^{(}}_{z^{(}}$$

$$\frac{\partial \mathscr{L}_n}{\partial w_{i,j}^{(l)}} \quad ?$$

Chain rule

$$\mathscr{L}_n = \frac{1}{2}\left(y_n - f^{(L+1)} \circ \cdots \circ f^{(l+1)} \circ \phi\left(z^{(l)}\right)\right)^2$$

Apply the chain rule:

$$\frac{\partial \mathscr{L}_n}{\partial w_{i,j}^{(l)}} = \sum_{k=1}^K \frac{\partial \mathscr{L}_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{i,j}^{(l)}}$$

$$= \frac{\partial \mathscr{L}_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l)}} \quad \text{since } \frac{\partial z_k^{(l)}}{\partial w_{i,j}^{(l)}} = 0 \text{ for } k \neq j$$

$$= \frac{\partial \mathscr{L}_n}{\partial z_j^{(l)}} \cdot x_i^{(l-1)} \quad \text{since } z_j^{(l)} = \sum_{k=1}^K w_{k,j}^{(l)} x_k^{(l-1)}$$

We need to compute $\frac{\partial \mathscr{L}_n}{\partial z_j^{(l)}}, z^{(l)}, x_i^{(l-1)}$ and reuse them for different $\frac{\partial \mathscr{L}_n}{\partial w_{i,j}^{(l)}}$

Forward Pass

We can compute $z^{(l)}$ and $x^{(l)}$ by a forward pass in the network:

$$x^{(0)} = x_n \in \mathbb{R}^d$$
$$z^{(l)} = \left(\mathbf{W}^{(l)}\right)^\top x^{(l-1)} + b^{(l)}$$
$$x^{(l)} = \phi\left(z^{(l)}\right)$$

Computational complexity:
$\Rightarrow$ one pass over the network $O\left(K^2 L\right)$

Backward pass (I)

Define $\delta_j^{(l)} = \frac{\partial \mathscr{L}_n}{\partial z_j^{(l)}}$

Chain rule:

$$\delta_j^{(l)} = \frac{\partial \mathscr{L}_n}{\partial z_j^{(l)}} = \sum_k \frac{\partial \mathscr{L}_n}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_k \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

Backward pass (II)

Using $z_k^{(l+1)} = \sum_{i=1}^K w_{i,k}^{(l+1)} x_i^{(l)} + b_k^{(l+1)} = \sum_{i=1}^K w_{i,k}^{(l+1)} \phi\left(z_i^{(l)}\right) + b_k^{(l+1)}$

We obtain $\frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \phi'\left(z_j^{(l)}\right) w_{j,k}^{(l+1)}$

Thus

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} \phi'\left(z_j^{(l)}\right) w_{j,k}^{(l+1)}$$

It can be written in vector form:

$$\delta^{(l)} = \left(\mathbf{W}^{(l+1)} \delta^{(l+1)}\right) \odot \phi'\left(z^{(l)}\right)$$

Backward pass (III)

Initialization:

$$\delta^{(L+1)} = \frac{\partial}{\partial z^{(L+1)}} \frac{1}{2}\left(y_n - z^{(L+1)}\right)^2 = z^{(L+1)} - y_n$$

Compute all the $\delta^{(l)}$ by a backward pass in the network:

$$\delta^{(l)} = \left(\mathbf{W}^{(l+1)} \delta^{(l+1)}\right) \odot \phi'\left(z^{(l)}\right)$$

Computational complexity: one pass over the network $O\left(K^2 L\right)$

Derivatives computation

Using that $z_m^{(l)} = \sum_{k=1}^K w_{k,m}^{(l)} x_k^{(l-1)} + b_m^{(l)}$:

- $\frac{\partial \mathscr{L}_n}{\partial b_j^{(l)}} = \sum_{k=1}^K \frac{\partial \mathscr{L}_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial b_j^{(l)}} = \frac{\partial \mathscr{L}_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)}$
- $\frac{\partial \mathscr{L}_n}{\partial w_{i,j}^{(l)}} = \sum_{k=1}^K \frac{\partial \mathscr{L}_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial \mathscr{L}_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l)}} = \delta_j^{(l)} \cdot x_i^{(l-1)}$

Backpropagation algorithm

Forward pass:

$$x^{(0)} = x_n \in \mathbb{R}^d$$
$$z^{(l)} = \left(\mathbf{W}^{(l)}\right)^\top x^{(l-1)} + b^{(l)}$$
$$x^{(l)} = \phi\left(z^{(l)}\right)$$

Backward pass:

$$\delta^{(L+1)} = z^{(L+1)} - y_n$$
$$\delta^{(l)} = \left(\mathbf{W}^{(l+1)} \delta^{(l+1)}\right) \odot \phi'\left(z^{(l)}\right)$$

Compute the derivatives:

$$\frac{\partial \mathscr{L}_n}{\partial w_{i,j}^{(l)}} = \delta_j^{(l)} x_i^{(l-1)}$$
$$\frac{\partial \mathscr{L}_n}{\partial b_j^{(l)}} = \delta_j^{(l)}$$

Parameter Initialization

Importance of Parameter Initialization

- In deep networks, improper parameter initialization can lead to the vanishing or exploding gradients problem
- Problem: Extremely slow or unstable optimization
- Solution: Control the layerwise variance of neurons (aka He initialization)
- Note: As illustrated, even a two-fold difference in the scale of

Source: Delving Deep into Rectifiers: Surpassing Human-Level
Performance on ImageNet Classification (CVPR 2015) initialization can be crucial

Variance-Preserving Initialization

Variance-preserving initialization for ReLU networks:

- $z^{(l)} \sim \mathcal{N}\left(0, \mathbf{I}_K\right)$: pre-activations at layer l (note: $\operatorname{Var}\left[z_i^{(l)}\right] = 1$)
- $w_i^{(l+1)} \sim \mathcal{N}\left(0, \sigma \mathbf{I}_K\right)$: the i-th weight vector at layer $l+1$
- $z_i^{(l+1)} = \operatorname{ReLU}\left(z^{(l)}\right)^\top w_i^{(l+1)}$: the i-th pre-activation at layer $l+1$

Question: How should we set σ so that $\operatorname{Var}\left[z_i^{(l+1)}\right] = 1$?

Answer: $\sigma = \sqrt{2/K}$

Derivation: Refer to the exercise for the derivation

Normalization Layers

Batch Normalization

Consider a batch $B = (x_1, \cdots, x_M)$ and denote by $z_n^{(l)}$ the layer's pre-activation input corresponding to the observation x_n

Batch Normalization

Consider a batch $B = (x_1, \cdots, x_M)$ and denote by $z_n^{(l)}$ the layer's pre-activation input corresponding to the observation x_n

One input x_n

Batch Normalization

Consider a batch $B = (x_1, \cdots, x_M)$ and denote by $z_n^{(l)}$ the layer's pre-activation input corresponding to the observation x_n

Batch Normalization

Consider a batch $B = (x_1, \cdots, x_M)$ and denote by $z_n^{(l)}$ the layer's pre-activation input corresponding to the observation x_n

Step 1: Normalize each layer's input using its mean and its variance over the batch:

$$\bar{z}_n^{(l)} = \frac{z_n^{(l)} - \mu_B^{(l)}}{\sqrt{\left(\sigma_B^{(l)}\right)^2 + \varepsilon}}$$

Component-wise

where $\mu_B^{(l)} = \frac{1}{M} \sum_{n=1}^M z_n^{(l)}$ and $\left(\sigma_B^{(l)}\right)^2 = \frac{1}{M} \sum_{n=1}^M \left(z_n^{(l)} - \mu_B^{(l)}\right)^2$, and $\varepsilon \in \mathbb{R}_{\geq 0}$ is a small value added for numerical stability

Step 2: Introduce learnable parameters $\gamma^{(l)}, \beta^{(l)} \in \mathbb{R}^K$ to be able to reverse the normalization if needed:

$$\hat{z}_n^{(l)} = \gamma^{(l)} \odot \bar{z}_n^{(l)} + \beta^{(l)}$$

Batch Normalization

Scale-invariance: For $\varepsilon \approx 0$, the output is invariant to activation-wise affine scaling of $z_n^{(l)}$

$$\operatorname{BN}\left(a \odot z_n^{(l)} + b\right) = \operatorname{BN}\left(z_n^{(l)}\right) \quad \text{for } a \in \mathbb{R}_{>0}^K \text{ and } b \in \mathbb{R}^K$$

Thus, for example, there is no need to include a bias before BatchNorm.

Inference: The prediction for one sample should not depend on other samples

- Estimate $\hat{\mu}^{(l)} = \mathbf{E}\left[\mu_B^{(l)}\right]$ and $\hat{\sigma}^{(l)} = \mathbf{E}\left[\sigma_B^{(l)}\right]$ during training, use these for inference
- Exponential moving averages are commonly used in practice

Implementation:

- Requires sufficiently large batches to get good estimates of $\mu_B^{(l)}, \sigma_B^{(l)}$
- BatchNorm is applied a bit differently for non-fully-connected nets (see the pytorch docs for CNNs)
- In PyTorch, switch modes by using model.train() for training and model.eval() for inference

Batch Normalization - Results

Source: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (ICML 2015)

- BatchNorm leads to much faster convergence
- BatchNorm allows to use much larger learning rates (up to 30×)

Layer Normalization

Step 1: Normalize each layer's input using its mean and its variance over the features (instead of over the inputs):

$$\bar{z}_n^{(l)} = \frac{z_n^{(l)} - \mu_n^{(l)} \cdot 1_K}{\sqrt{\left(\sigma_n^{(l)}\right)^2 + \varepsilon}}$$

where $\mu_n^{(l)} = \frac{1}{K} \sum_{k=1}^K z_n^{(l)}(k)$ and $\left(\sigma_n^{(l)}\right)^2 = \frac{1}{K} \sum_{k=1}^K \left(z_n^{(l)}(k) - \mu_n^{(l)}\right)^2$, and $\varepsilon \in \mathbb{R}_{\geq 0}$

Step 2: Introduce learnable parameters $\gamma^{(l)}, \beta^{(l)} \in \mathbb{R}^K$:

Remarks:

$$\hat{z}_n^{(l)} = \gamma^{(l)} \odot \bar{z}_n^{(l)} + \beta^{(l)}$$

- Normalize across features, independently for each observation
- Very common alternative, widely used for transformers and text data
- No batch dependency, use the same for training and inference

Normalization - conclusion

Benefits of normalization layers:

- Stabilizes activation magnitudes / reduces initialization impact
- Additional regularization effect from noisy $\mu_B^{(l)}, \sigma_B^{(l)}$ in batch norm
- Stabilizes and speeds up training, allows larger learning rates

Used in almost all modern deep learning architectures

- Often inserted after every convolutional layer, before non-linearity

Recap

- Neural networks are trained with gradient-based methods such as SGD
- To compute the gradients, we use backpropagation, which involves the chain rule to efficiently calculate the gradients based on the network's intermediate outputs $z^{(l)}$ and $\delta^{(l)}$
- Proper parameter initialization should avoid exploding and vanishing gradients by carefully controlling the layerwise variance
- Batch and Layer normalization dynamically stabilize the training process, allowing for faster convergence and the use of larger learning rates

---

Transformers

- A transformer is a neural network that iteratively transforms a sequence to another sequence and mixes the information between the sequence elements via self-attention.

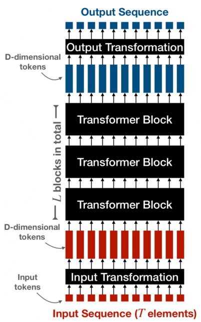
Architecture

- Self-Attention (SA): mixes information between tokens

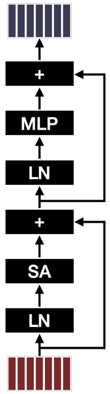
- Multi-Layer Perceptron (MLP): mixes information within each token

- Skip connections are widely used

- Layer normalization (LN) is usually placed at the start of a residual branch



Text Token Embeddings



- Tokenization: split the input text into a sequence of input tokens (typically word fragments + some special symbols) according to some predefined tokenizer procedure:

- Convert each token ID $i \in \{1, ..., N_{vocab}\}$ into a real-valued vector $\mathbf{w}_i \in \mathbb{R}^D$

- This can be seen as a matrix multiplication $\mathbf{W} \cdot \mathbf{e}_i = \mathbf{W}_{:,i} = \mathbf{w}_i$ (with $\mathbf{W} \in \mathbb{R}^{D \times N_{vocab}}$)

- $\mathbf{W}$ is learned via backpropagation, along with all other transformer parameters (however, the tokenizer procedure is typically fixed in advance and not learned)

- The whole input sequence of T tokens leads to an input matrix $X \in \mathbb{R}^{T \times D}$

Attention

- Attention is a function that transforms a sequence of tokens to a new sequence of tokens using a learned input-dependent weighted average

- Input tokens : $V \in \mathbb{R}^{T_{in} \times D}$

- Output tokens : $Z \in \mathbb{R}^{T_{out} \times D}$

- Output tokens are simply a weighted average of the input tokens: $z_i = \sum_{j=1}^{T_i} p_{ij} v_j$ i.e. $Z = PV$

- Weighting coefficients $\mathcal{P} \in [0,1]^{T_{out} \times T_{in}}$ form valid probability distributions over the input tokens $\sum_{j=1}^{T_{in}} p_{ij} = 1$

- Query tokens : $Q \in \mathbb{R}^{T_{out} \times D_K}$

- Key tokens : $K \in \mathbb{R}^{T_{in} \times D_K}$

- Determine weight $p_{i,j}$ based on how simmilar q_i and k_j are.

- Use inner product to obtain raw similarity scores.

- Normalize with softmax (scaled the temperature by $\sqrt{D_K}$) to obtain a probability distribution.

- $P = \text{softmax}\left(\frac{QK^\top}{\sqrt{D_K}}\right)$ The softmax is applied on each row independently. Scaling ensures uniformity at initialization and faster convergence

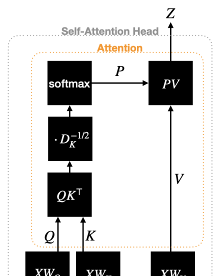
Self-Attention

- V, K, Q are all derived from the same input token sequence $X \in \mathbb{R}^{T \times D}$

- Values : $V = XW_V \in \mathbb{R}^{T \times D}$, $W_V \in \mathbb{R}^{D \times D}$

- Keys : $K = XW_K \in \mathbb{R}^{T \times D_K}$, $W_K \in \mathbb{R}^{D \times D_K}$

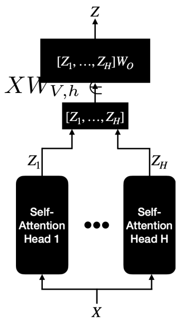
- Queries : $Q = XW_Q \in \mathbb{R}^{T \times D_K}$, $W_Q \in \mathbb{R}^{D \times D_K}$



- W_Q, W_V, W_K are learned parameters.

- $Z = \text{softmax}\left(\frac{XW_Q W_K^T X^T}{\sqrt{D_K}}\right) XW_V$

Multi-Head Self-Attention



- Run H Self-Attention "heads" in parallel

- $Z_h = \text{softmax}\left(\frac{XW_{Q,h}W_{K,h}^\top X^\top}{\sqrt{D_K}}\right) XW_{V,h} \in \mathbb{R}^{T \times D_V}$

- $W_{V,h} \in \mathbb{R}^{D \times D_V}$, $W_{K,h} \in \mathbb{R}^{D \times D_K}$, $W_{Q,h} \in \mathbb{R}^{D \times D_K}$

- The final output is obtained by concatenating head-outputs and applying a linear transformation $Z = [Z_1, \ldots, Z_H]W_o$ where $W_O \in \mathbb{R}^{HD_V \times D}$ is learned via backpropagation

Positional Information

- Attention by itself does not account for the order of input

- incorporate a positional encoding in the network which is a function from the position to a feature vector $pos : \{1, ..., T\} \to \mathbb{R}^D$

- The most basic choice is to add a positional embedding W_{pos} corresponding to each token's position t to the input embedding. $W_{pos} \in \mathbb{R}^{D \times T}$ is learned via backpropagation along with the other parameters

MLP

- Mixing Information within Tokens

- Apply the same transformation to each token independently: $MLP(X) = \varphi(XW_1)W_2$

- $W_1, W_2 \in \mathbb{R}^{D \times D}$ learned via backprop

Output Transformations

- typically simple: linear transformation or a small MLP

- dependent on the task: Single output (e.g.,sequence-level classification): apply an output transformation to a special taskspecific input token or to the average of all tokens. Multiple outputs (e.g., per-token classification): apply an output transformation to each token independently

Vision Transformer Architecture

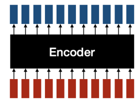
- Self-attention is more general than convolution and can potentially express it

- The receptive field is the whole image after just one self-attention layer

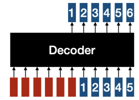
- ViTs require more data than CNNs due to their reduced inductive bias in extracting local features

- In many cases, the model attends to image regions that are semantically relevant for classification

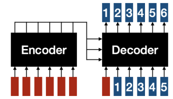
Encoders & Decoders



- Encoders (e.g., classification): They produce a fixed output size and process all inputs simultaneously



- Decoders (e.g., ChatGPT): Auto-regressively sample the next token as $x_{t+1} \sim sofimax(f(x_1, ..., x_t))$ and use it as new input token. Capable of generating responses of arbitrary length.



- Encoder-decoder (e.g., translation): First encode the whole input (e.g., in one language) and then decode to token by token (e.g., in a different language)

Adversarial ML

- We don't understand how NN models generalize and react to shifts in the distribution of data (i.e., distribution shifts)

- Classification problem: $(X, Y) \sim \mathcal{D}$, Y with range $\{-1, 1\}$

- Standard risk: average zero-one loss over X: $R(f) = \mathbb{E}_{\mathcal{D}}\left[1_{f(X) \neq Y}\right] = \mathbb{P}_{\mathcal{D}}\left[f(X) \neq Y\right]$ i.e. minimise proba of wrong prediction.

- Adversarial risk: average zero-one loss over small, worst-case perturbations of X: $R_\varepsilon(f) = \mathbb{E}_{\mathcal{D}}\left[\max_{\hat{x}, \|\hat{x} - X\| \le \varepsilon} 1_{f(\hat{x}) \neq Y}\right]$

Generating adversarial examples

- Task: given an input (x, y) and a model $f : \mathcal{X} \to \{-1, 1\}$ find an input $\hat{x}$ s.t.: a) $\|\hat{x} - x\| \le \varepsilon$ b) the model f makes a mistake on it.

- Trivial case: x already missclassified $\to$ no action required

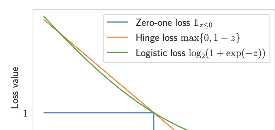
- General case: find $\hat{x}$ such that at $f(\hat{x}) \neq y$ and $\left||\hat{x} - x|\right| \le \varepsilon$ i.e. $\hat{x} \in B_x(\varepsilon) \cap \{x' f(x') = -y\}$

- Optimization problem with respect to the inputs

- Problem: optimizing the indicator function is difficult: 1) The indicator function 1 is not continuous 2) The NN prediction f outputs discrete class values $\{-1, 1\}$

- Replace the difficult problem involving the indicator with a smooth problem $\max_{\hat{x}, \|\hat{x} - X\| \le \varepsilon} 1_{f(\hat{x}) \neq Y} \to \max_{\hat{x}, \|\hat{x} - X\| \le \varepsilon} \ell(yg(\hat{x}))$

- decreasing, margin-based (i.e., dependent on $y * g(x)$) classification losses



White-Box attacks

- Solve $\max_{\hat{x},\|\hat{x}-X\|\le\varepsilon} \ell(yg(\hat{x}))$ knowing g

- $\nabla_x \ell(yg(x)) = y\ell'(yg(x))\nabla_x g(x)$, with $y\ell'(yg(x)) \le 0$ since classification losses are decreasing.

- Move in direction of $\propto -y\,\nabla_x g(x)$

- Interpretation $f(x) = \text{sign}(g(x))$: If $y=1$ we want to decrease $g(x)$ and follow $-\nabla_x g(x)$. If $y=-1$ we want to decrease $g(x)$ and follow $\nabla_x g(x)$

- By using ℓ and not directly $yg(\hat{x})$ it will extend to multi-class classification and robust training.

- linearize the loss $\tilde{\ell}(x) := \ell(yg(x))$

$\max_{\|\hat{x}-x\|\le\varepsilon} \tilde{\ell}(x)$

$\approx \max_{\|\hat{x}-x\|\le\varepsilon} \tilde{\ell}(x) + \nabla_x\tilde{\ell}(x)^T(\hat{x}-x)$

$= \tilde{\ell}(x) + \max_{\|\hat{x}-x\|\le\varepsilon} \nabla_x\tilde{\ell}(x)^T(\hat{x}-x)$

$= \tilde{\ell}(x) + \max_{\|\delta\|\le\varepsilon} \nabla_x\tilde{\ell}(x)^T\delta$

- We need to maximize the inner product under a norm constraint, i.e. find the optimal local update

- This is a simple problem for which we can get a closed-form solution depending on the norm used to measure the perturbation size $||\delta||$

One-step attack

- Solution for the ℓ_2 norm:

$\delta_2^\star = \varepsilon \cdot \frac{\nabla_x\tilde{\ell}(x)}{||\nabla_x\tilde{\ell}(x)||_2} = -\,\varepsilon y * \frac{\nabla_x g(x)}{||\nabla_x g(x)||_2} \Rightarrow$

$\hat{x} = x - \varepsilon y \cdot \frac{\nabla_x g(x)}{\|\nabla_x g(x)\|_2}$

- Solution for the ℓ_∞ norm called **Fast Gradient Sign Method**:

$\delta_\infty^\star = \varepsilon \cdot \text{sign}(\nabla_x\tilde{\ell}(x)) = -\,\varepsilon y \cdot \text{sign}(\nabla_x g(x)) \Rightarrow$

$\hat{x} = x - \varepsilon y \cdot \text{sign}(\nabla_x g(x))$

Multi-step attack

- These updates can be done iteratively and combined with a projection Π on the feasible set (i.e., balls ℓ_2 / ℓ_∞ here)

- Projected Gradient Descent (PGD attack)

- ℓ_2 norm:

$\delta^{t+1} = \Pi_{B_2(e)}[\delta^t + \alpha \cdot \frac{\nabla\tilde{\ell}(x+\delta^t)}{\|\nabla\tilde{\ell}(x+\delta^t)\|_2}]$

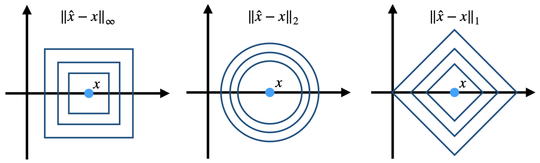
$$\Pi_{B_2(\varepsilon)}(\delta) = \begin{cases} \varepsilon\cdot\delta/\|\delta\|_2, & \text{if } \|\delta\|_2 \ge \varepsilon \\ \delta \text{ otherwise} \end{cases}$$

- ℓ_∞ norm:

$$\delta^{t+1} = \Pi_{B_\infty(\varepsilon)}\left[\delta^t + \alpha\cdot\text{sign}(\nabla\tilde{\ell}(x+\delta^t))\right],$$

$$\Pi_{B_\infty(\varepsilon)}(\delta)_i = \begin{cases} \varepsilon\cdot\text{sign}(\delta_i), & \text{if } |\delta_i| \ge \varepsilon \\ \delta_i \text{ otherwise} \end{cases}$$

- the gradients are computed by backprop w.r.t. inputs, not parameters!



Black-box attacks

- We don't know $g(x)$

- Obtaining a surrogate model can be costly and there is no guarantee of success

- Query-based methods often require a lot of queries (10k-100k), easy to restrict access for the attacker!

Query-based gradient estimation

- Score-based: we can query the continuous model scores $g(x)\in\mathbb{R}$. We can approximate the gradient by using the finite difference formula:

$\nabla_x g(x) \approx \sum_{i=1}^d \frac{g(x+\alpha e_i)-g(x)}{\alpha} e_i$

- Decision-based: we can query only the predicted class $f(x)\in\{-1,1\}$, similar techniques can be adapted for the decision-based case.

Transfer Attacks

- Train a similar surrogate model $\hat{f}\approx f$ on similar data

- Model stealing (query f given some unlabeled inputs $\{x_n, f(x_n)\}_{n=1}^N$) can facilitate transfer attacks.

Adversarial training

- Adversarial training: the goal is to minimize the adversarial risk:

$\min_\theta R_\varepsilon(f_\theta) = \mathbb{E}_{\mathcal{D}}\left[\max_{\hat{x},\|\hat{x}-X\|\le\varepsilon} 1_{f(\hat{x})\ne Y}\right]$

- $\mathcal{D}$ unknown $\to$ approximate it with a sample average + classification loss is non-continuous $\to$ use a smooth loss $\Rightarrow$

$\min_\theta \frac{1}{N}\sum_{n=1}^N \max_{\hat{x}_n,\|x_n-\hat{x}_n\|\le\varepsilon} \ell(y_n g_\theta(\hat{x}_n))$

1) $\forall x_n$, $\hat{x}_n^\star \approx \arg\max_{||x_n-\hat{x}_n|\le\varepsilon} \ell(y_n g_\theta(\hat{x}_n))$

2) GD step w.r.t. θ using $\frac{1}{N}\sum_{n=1}^N \nabla_\theta \ell(y_n g_\theta(\hat{x}_n^\star))$

Advantages

- state-of-the-art approach for robust classification

- more interpretable gradients

- fully compatible with SGD

Disadvantages

- Increased computational time: proportional to the number of PGD steps

- Robustness-accuracy tradeoff: using too large ε leads to worse standard accuracy

Adversarial Example

$x\in\mathbb{R}^d, y \sim Bernoulli(\{-1,1\}), Z_i \sim \mathcal{N}(0,1)$

- Robust features: $x_1 = y + Z_1$

- Non-robust features: $x_i = y\sqrt{\frac{\log d}{d-1}} + Z_i$, $\forall i \in \{-1,1\}$

- $d\to\infty \Rightarrow \uparrow$ adversarial risk and $\downarrow$ standard risk

- using the robust feature x_1:

MLE: $\arg\max_{\hat{y}\in\{\pm1\}} p(\hat{y}\mid x_1) =$

$\arg\max_{\hat{y}\in\{\pm1\}} \frac{p(x_1|\hat{y})p(\hat{y})}{p(x_1)} = \arg\max_{\hat{y}\in\{\pm1\}} p(x_1 \mid \hat{y}$

assuming $p(y=1) = p(y=-1)$

- Standard Risk: $\int_0^\infty \frac{1}{\sqrt{2\pi}} e^{-0.5(x+1)^2} dx \approx 0.16$ good but not perfect!

- using both robust and non-robust features:

MLE for all features $x_i = ya_i + Z_i$

$\arg\max_{\hat{y}\in\{\pm1\}} p(\hat{y}\mid x)$

$= \arg\max_{\hat{y}\in\{\pm1\}} \prod_{i=1}^d p(x_i\mid\hat{y})$

$= \arg\max_{\hat{y}\in\{\pm1\}} \sum_{i=1}^d \log p(x_i\mid\hat{y})$

$= \arg\max_{\hat{y}\in\{\pm1\}} \sum_{i=1}^d \log \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x_i-\hat{y}a_i)^2}$

$= \arg\min_{\hat{y}\in\{\pm1\}} \sum_{i=1}^d (x_i-\hat{y}a_i)^2$

$= \arg\min_{\hat{y}\in\{\pm1\}} \sum_{i=1}^d (x_i^2 - 2x_i\hat{y}a_i + \hat{y}^2a_i^2)$

$= \arg\max_{\hat{y}\in\{\pm1\}} \hat{y}\sum_{i=1}^d x_ia_i$

$\hat{y}\sum_{i=1}^d x_ia_i = \hat{y}y(\sum_{i=1}^d a_i^2) + \hat{y}\sum_{i=1}^d a_iZ_i = \hat{y}y(1+\log(d)) + \hat{y}Z$ where $Z := \sum_{i=1}^d a_iZ_i \sim \mathcal{N}(0, 1+\log d)$

Scaling by $1/(1+\log d)$ the MLE results in:

$y\hat{y} + \hat{y}Z$ with $Z\sim\mathcal{N}(0, 1/(1+\log d))$

$d\to\infty, \hat{y}Z \to 0 \Rightarrow$ standard risk $R(f)\to 0$

- using the non-robust features improves standard risk!

- Adversarial risk:

The adversary can use tiny ℓ_∞ perturbations:

$\varepsilon = 2\sqrt{\frac{\log d}{d-1}}$ ($\to 0$ when) $d\to\infty$)

$\hat{x}_1 = \left(1 - 2\sqrt{\frac{\log d}{d-1}}\right) y + Z_1$, almost unaffected

$\hat{x}_i = -\sqrt{\frac{\log d}{d-1}} y + Z_i$, completely flipped

$R_\varepsilon(f) \approx 1 \Rightarrow$ tradeoff between accuracy and robustness.

Clustering

Clusters are groups of points whose inter-point distances are small compared to the distances outside the cluster.

The goal is to find "prototype" points $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$ and cluster assignments $z_n \in \{1,2,\ldots,K\}$ for all $n = 1,2,\ldots,N$ data vectors $\mathbf{x}_n \in \mathbb{R}^D$.

K-means clustering

Assume K is known.

$$\min_{\mathbf{z},\boldsymbol{\mu}} \mathcal{L}(\mathbf{z},\boldsymbol{\mu}) = \sum_{n=1}^N \sum_{k=1}^K z_{nk}\|\mathbf{x}_n - \boldsymbol{\mu}_k\|_2^2$$

s.t. $\boldsymbol{\mu}_k \in \mathbb{R}^D, z_{nk}\in\{0,1\}, \sum_{k=1}^K z_{nk} = 1$,

where $\mathbf{z}_n = [z_{n1}, z_{n2}, \ldots, z_{nK}]^\top$

$$\mathbf{z} = [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N]^\top$$

$$\boldsymbol{\mu} = [\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K]^\top$$

Is this optimization problem easy?

Algorithm: Initialize $\boldsymbol{\mu}_k \forall k$, then iterate:

1. For all n, compute $\mathbf{z}_n$ given $\boldsymbol{\mu}$.
2. For all k, compute $\boldsymbol{\mu}_k$ given $\mathbf{z}$.

Step 1: For all n, compute $\mathbf{z}_n$ given $\boldsymbol{\mu}$.

$$z_{nk} = \begin{cases} 1 \text{ if } k = \arg\min_{j=1,2,\ldots K} \|\mathbf{x}_n - \boldsymbol{\mu}_j\|_2^2 \\ 0 \text{ otherwise} \end{cases}$$

Step 2: For all k, compute $\boldsymbol{\mu}_k$ given $\mathbf{z}$.

Take derivative w.r.t. $\boldsymbol{\mu}_k$ to get:

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^N z_{nk}\mathbf{x}_n}{\sum_{n=1}^N z_{nk}}$$

Hence, the name 'K-means'.

Summary of K-means

Initialize $\boldsymbol{\mu}_k \forall k$, then iterate:

1. For all n, compute $\mathbf{z}_n$ given $\boldsymbol{\mu}$.

$$z_{nk} = \begin{cases} 1 \text{ if } k = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|_2^2 \\ 0 \text{ otherwise} \end{cases}$$

2. For all k, compute $\boldsymbol{\mu}_k$ given $\mathbf{z}$.

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^N z_{nk}\mathbf{x}_n}{\sum_{n=1}^N z_{nk}}$$

Convergence to a local optimum is assured since each step decreases the cost (see Bishop, Exercise 9.1).

Coordinate descent

K-means is a coordinate descent algorithm, where, to find $\min_{\mathbf{z},\boldsymbol{\mu}} \mathcal{L}(\mathbf{z},\boldsymbol{\mu})$, we start with some $\boldsymbol{\mu}^{(0)}$ and repeat the following:

$\mathbf{z}^{(t+1)} := \arg\min_{\mathbf{z}} \mathcal{L}\left(\mathbf{z}, \boldsymbol{\mu}^{(t)}\right)$

$\boldsymbol{\mu}^{(t+1)} := \arg\min_{\boldsymbol{\mu}} \mathcal{L}\left(\mathbf{z}^{(t+1)}, \boldsymbol{\mu}\right)$

Examples

K-means for the "old-faithful" dataset (Bishop's Figure 9.1)

(e) Iteration 0
(h) Iteration 2
(k) Iteration 3
(f) Iteration 1
(i) Iteration 2
(l) Iteration 4
(g) Iteration 1
(j) Iteration 3
(m) Iteration 4

Data compression for images (this is also known as vector quantization).

Probabilistic model for K-means

K-means as a Matrix Factorization

Recall the objective

$$\min_{\mathbf{z},\boldsymbol{\mu}} \mathcal{L}(\mathbf{z},\boldsymbol{\mu}) = \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}\left\|\mathbf{x}_n - \boldsymbol{\mu}_k\right\|_2^2$$
$$= \left\|\mathbf{X}^\top - \mathbf{M}\mathbf{Z}^\top\right\|_{\text{Frob}}^2$$

s.t. $\boldsymbol{\mu}_k \in \mathbb{R}^D$,

$$z_{nk} \in \{0,1\}, \sum_{k=1}^{K} z_{nk} = 1$$

Issues with K-means

1. Computation can be heavy for large N, D and K.
2. Clusters are forced to be spherical (e.g. cannot be elliptical).
3. Each example can belong to only one cluster ("hard" cluster assignments).

Motivation

K-means forces the clusters to be spherical, but sometimes it is desirable to have elliptical clusters. Another issue is that, in K-means, each example can only belong to one cluster, but this may not always be a good choice, e.g. for data points that are near the "border". Both of these problems are solved by using Gaussian Mixture Models.

Clustering with Gaussians

The first issue is resolved by using full covariance matrices $\boldsymbol{\Sigma}_k$ instead of isotropic covariances.
$p(\mathbf{X} \mid \boldsymbol{\mu},\boldsymbol{\Sigma},\mathbf{z}) = \prod_{n=1}^{N}\prod_{k=1}^{K}\left[\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right]^{z_{nk}}$

Soft-clustering

The second issue is resolved by defining z_n to be a random variable. Specifically, define $z_n \in \{1,2,\ldots,K\}$ that follows a multinomial distribution.

$p\left(z_n = k\right) = \pi_k$ where $\pi_k > 0, \forall k$ and $\sum_{k=1}^{K}\pi_k = 1$

This leads to soft-clustering as opposed to having "hard" assignments.

Gaussian mixture model

Together, the likelihood and the prior define the joint distribution of Gaussian mixture model (GMM):

$$p(\mathbf{X},\mathbf{z} \mid \boldsymbol{\mu},\boldsymbol{\Sigma},\boldsymbol{\pi})$$
$$= \prod_{n=1}^{N} p\left(\mathbf{x}_n \mid z_n,\boldsymbol{\mu},\boldsymbol{\Sigma}\right) p\left(z_n \mid \boldsymbol{\pi}\right)$$
$$= \prod_{n=1}^{N}\prod_{k=1}^{K}\left[\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right]^{z_{nk}} \prod_{k=1}^{K}\left[\pi_k\right]^{z_{nk}}$$

Here, $\mathbf{x}_n$ are observed data vectors, z_n are latent unobserved variables, and the unknown *pa* rameters are given by $\boldsymbol{\theta} := \{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\boldsymbol{\Sigma}_1,\ldots,\boldsymbol{\Sigma}_K,\boldsymbol{\pi}\}$.

Marginal likelihood

GMM is a latent variable model with z_n being the unobserved (latent) variables. An advantage of treating z_n as latent variables instead of parameters is that we can marginalize them out to get a cost function that does not depend on z_n, i.e. as if z_n never existed.
Specifically, we get the following marginal likelihood by marginalizing z_n out from the likelihood:
$p\left(\mathbf{x}_n \mid \boldsymbol{\theta}\right) = \sum_{k=1}^{K}\pi_k\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)$
Deriving cost functions this way is good for statistical efficiency. Without a latent variable model, the number of parameters grows at rate $\mathcal{O}(N)$. After marginalization, the growth is reduced to $\mathcal{O}\left(D^2K\right)$ (assuming $D, K \ll N$).

Maximum likelihood

To get a maximum (marginal) likelihood estimate of $\boldsymbol{\theta}$, we maximize the following:

$$\max_{\boldsymbol{\theta}} \sum_{n=1}^{N}\log\sum_{k=1}^{K}\pi_k\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)$$

Is this cost convex? Identifiable? Bounded?

Motivation

Computing maximum likelihood for Gaussian mixture model is difficult due to the log outside the sum.
$\max_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) := \sum_{n=1}^{N}\log\sum_{k=1}^{K}\pi_k\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)$
Expectation-Maximization (EM) algorithm provides an elegant and general method to optimize such optimization problems. It uses an iterative two-step procedure where individual steps usually involve problems that are easy to optimize.

EM algorithm: Summary

Start with $\boldsymbol{\theta}^{(1)}$ and iterate:

1. Expectation step: Compute a lower bound to the cost such that it is tight at the previous $\boldsymbol{\theta}^{(t)}$:
$\mathcal{L}(\boldsymbol{\theta}) \geq \underline{\mathcal{L}}\left(\boldsymbol{\theta},\boldsymbol{\theta}^{(t)}\right)$ and
$\mathcal{L}\left(\boldsymbol{\theta}^{(t)}\right) = \underline{\mathcal{L}}\left(\boldsymbol{\theta}^{(t)},\boldsymbol{\theta}^{(t)}\right)$.
2. Maximization step: Update $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}}\underline{\mathcal{L}}\left(\boldsymbol{\theta},\boldsymbol{\theta}^{(t)}\right).$$

Concavity of log

Given non-negative weights q s.t. $\sum_k q_k = 1$, the following holds for any $r_k > 0$:
$\log\left(\sum_{k=1}^{K} q_k r_k\right) \geq \sum_{k=1}^{K} q_k \log r_k$

The expectation step

$\log\sum_{k=1}^{K}\pi_k\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right) \geq \sum_{k=1}^{K} q_{kn}\log\frac{\pi_k\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)}{q_{kn}}$
with equality when,
$q_{kn} = \frac{\pi_k\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)}{\sum_{k=1}^{K}\pi_k\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)}$
This is not a coincidence.

The maximization step

Maximize the lower bound w.r.t. $\boldsymbol{\theta}$.
$\max_{\boldsymbol{\theta}}\sum_{n=1}^{N}\sum_{k=1}^{K} q_{kn}^{(t)}\left[\log\pi_k + \log\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right]$
Differentiating w.r.t. $\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k^{-1}$, we can get the updates for $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$.

$$\boldsymbol{\mu}_k^{(t+1)} := \frac{\sum_n q_{kn}^{(t)}\mathbf{x}_n}{\sum_n q_{kn}^{(t)}}$$
$$\boldsymbol{\Sigma}_k^{(t+1)} := \frac{\sum_n q_{kn}^{(t)}\left(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right)\left(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right)^\top}{\sum_n q_{kn}^{(t)}}$$

For π_k, we use the fact that they sum to 1. Therefore, we add a Lagrangian term, differentiate w.r.t. π_k and set to 0 , to get the following update:

$$\pi_k^{(t+1)} := \frac{1}{N}\sum_{n=1}^{N} q_{kn}^{(t)}$$

Summary of EM for GMM

Initialize $\boldsymbol{\mu}^{(1)},\boldsymbol{\Sigma}^{(1)},\boldsymbol{\pi}^{(1)}$ and iterate between the E and M step, until $\mathcal{L}(\boldsymbol{\theta})$ stabilizes.

1. E-step: Compute assignments $q_{kn}^{(t)}$:

$$q_{kn}^{(t)} := \frac{\pi_k^{(t)}\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k^{(t)},\boldsymbol{\Sigma}_k^{(t)}\right)}{\sum_{k=1}^{K}\pi_k^{(t)}\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k^{(t)},\boldsymbol{\Sigma}_k^{(t)}\right)}$$

2. Compute the marginal likelihood (cost).

$$\mathcal{L}\left(\boldsymbol{\theta}^{(t)}\right) = \sum_{n=1}^{N}\log\sum_{k=1}^{K}\pi_k^{(t)}\mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k^{(t)},\boldsymbol{\Sigma}_k^{(t)}\right)$$

3. M-step: Update $\boldsymbol{\mu}_k^{(t+1)},\boldsymbol{\Sigma}_k^{(t+1)},\pi_k^{(t+1)}$.

$$\boldsymbol{\mu}_k^{(t+1)} := \frac{\sum_n q_{kn}^{(t)}\mathbf{x}_n}{\sum_n q_{kn}^{(t)}}$$
$$\boldsymbol{\Sigma}_k^{(t+1)} := \frac{\sum_n q_{kn}^{(t)}\left(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right)\left(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right)^\top}{\sum_n q_{kn}^{(t)}}$$
$$\pi_k^{(t+1)} := \frac{1}{N}\sum_n q_{kn}^{(t)}$$

If we let the covariance be diagonal i.e. $\boldsymbol{\Sigma}_k := \sigma^2\mathbf{I}$, then EM algorithm is same as K-means as $\sigma^2 \to 0$.
Figure 1: EM algorithm for GMM

Posterior distribution

We now show that $q_{kn}^{(t)}$ is the posterior distribution of the latent variable, i.e. $q_{kn}^{(t)} = p\left(z_n = k \mid \mathbf{x}_n,\boldsymbol{\theta}^{(t)}\right)$
$p\left(\mathbf{x}_n, z_n \mid \boldsymbol{\theta}\right) = p\left(\mathbf{x}_n \mid z_n,\boldsymbol{\theta}\right)p\left(z_n \mid \boldsymbol{\theta}\right) = p\left(z_n \mid \mathbf{x}_n,\boldsymbol{\theta}\right)p\left(\mathbf{x}_n \mid \boldsymbol{\theta}\right)$

EM in general

Given a general joint distribution $p\left(\mathbf{x}_n, z_n \mid \boldsymbol{\theta}\right)$, the marginal likelihood can be lower bounded similarly:
The EM algorithm can be compactly written as follows:
$\boldsymbol{\theta}^{(t+1)} := \arg\max_{\boldsymbol{\theta}}\sum_{n=1}^{N}\mathbb{E}_{p\left(z_n \mid \mathbf{x}_n,\boldsymbol{\theta}^{(t)}\right)}\left[\log p\left(\mathbf{x}_n, z\right.\right.$
Another interpretation is that part of the data is missing, i.e. $\left(\mathbf{x}_n, z_n\right)$ is the "complete" data and z_n is missing. The EM algorithm averages over the "unobserved" part of the data.

Matrix Factorization

Given items (movies) $d = 1, 2, ..., D$ and users $n = 1, 2, ..., N$, we define X to be the $D \times N$ matrix containing all rating entries. That is, x_{dn} is the rating of n-th user for d-th item. Note that most ratings x_{dn} are missing, and our task is to predict them accurately.

Algorithm

$X \approx WZ^T$, $W \in \mathbb{R}^{D\times K}$, $Z \in \mathbb{R}^{N\times K}$ tall matrices
$K << N, D$
$\min_{W,Z}\ \mathcal{L}(W,Z) := \frac{1}{2}\sum_{(d,n)\in\mathbb{Q}}[x_{dn} - (WZ^T)_{dn}]^2$
- We hope to "explain" each rating x_{dn} by a numerical representation of the corresponding item

and user - in fact by the inner product of an item feature vector with the user feature vector.
- The set $\Omega \subseteq [D] \times [N]$ collects the indices of the observed ratings of the input matrix X.
- This cost is not jointly convex w.r.t. W and Z, nor identifiable as $(w^*, z^*) \Leftrightarrow (\beta w^*, \beta^{-1} z^*)$

Choosing K

- $\uparrow K \Rightarrow$ overfitting ($\Leftrightarrow \downarrow K \Rightarrow$ underfitting). For $K >> N, D \Rightarrow (W^*, {Z^*}^T) = (X, I) = (I, X)$

Regularization

$\frac{1}{2} \sum_{(d,n)\in\Omega} [x_{dn} - (WZ^T)_{dn}]^2 + \frac{\lambda_w}{2} ||W||^2_{\text{Frob}} + \frac{\lambda_z}{2} ||Z||^2_{\text{Frob}}$ $\lambda_w, \lambda_z \in \mathbb{R} > 0$

Stochastic Gradient Descent

$\mathcal{L} = \frac{1}{|\Omega|} \sum_{(d,n)\in\Omega} \underbrace{\frac{1}{2}[x_{dn} - (\mathbf{W}\mathbf{Z}^\mathsf{T})_{dn}]^2}_{f_{d,n}}$

For one fixed element (d, n) of the sum, we derive the gradient entry (d', k) for W:

$\frac{\partial}{\partial w_{d',k}} f_{d,n}(W, Z) \in \mathbb{R}^{D\times K} =$
$\begin{cases} -\left[x_{dn} - (\mathbf{W}\mathbf{Z}^T)_{dn}\right] z_{n,k} & \text{if } d' = d \\ 0 \text{ otherwise} \end{cases}$

$\frac{\partial}{\partial z_{n',k}} f_{d,n}(W, Z) \in \mathbb{R}^{N\times K} =$
$\begin{cases} -\left[x_{dn} - (\mathbf{W}\mathbf{Z}^T)_{dn}\right] w_{d,k} & \text{if } n' = n \\ 0 \text{ otherwise} \end{cases}$

- cost: $\Theta(K)$ which is cheap!

Alternating Least Squares

- No missing entries:
$\frac{1}{2} \sum_{d=1}^{D} \sum_{n=1}^{N} \left[x_{dn} - \left(W\mathbf{Z}^\mathsf{T}\right)_{dn}\right]^2$
$= \frac{1}{2} \|\mathbf{X} - \mathbf{W}\mathbf{Z}^\mathsf{T}\|^2_{Frob}$

- We first minimize w.r.t. Z for fixed W and then minimize W given Z (closed form solutions):
$Z^\mathsf{T} := (\mathsf{W}^\mathsf{T}\mathsf{W} + \lambda_z \mathsf{I}_K)^{-1} \mathsf{W}^\mathsf{T}\mathsf{X}$
$\mathbf{W}^\mathsf{T} := (\mathbf{Z}^\mathsf{T}\mathbf{Z} + \lambda_w \mathbf{I}_K)^{-1} \mathbf{Z}^\mathsf{T}\mathbf{X}^\mathsf{T}$

- Cost: need to invert a $K \times K$ matrix
- With missing entries: Can you derive the ALS updates for the more general setting, when only the ratings $(d, n) \in \Omega$ contribute to the cost, i.e.
$\frac{1}{2} \sum_{(d,n)\in\Omega} \left[x_{dn} - \left(\mathsf{W}\mathsf{Z}^\mathsf{T}\right)_{dn}\right]^2$
Compute the gradient with respect to each group of variables, and set to zero.

Text Representation

-Finding numerical representations for words is fundamental for all machine learning methods dealing with text data.
-Goal: For each word, find mapping (embedding) $w_i \mapsto \mathbf{w}_i \in \mathbb{R}^K$

Co-Occurence Matrix

-A big corpus of un-labeled text can be represented as the co-occurrence counts. n_{ij} := #contexts where word w_i occurs together with word w_j.
- Needs definition of Context e.g. document, paragraph, sentence, window and Vocabulary $\mathcal{V} := \{w_1, \ldots, w_D\}$
- For words $w_d = 1, 2, \ldots, D$ and context words $w_n = 1, 2, \ldots, N$, the co-occurence counts n_{ij} form a very sparse $D \times N$ matrix.

Learning Word-Representations

- Find a factorization of the cooccurence matrix!
- Typically uses log of the actual counts, i.e. $x_{dn} := \log(n_{dn})$.
- Aim to find $\mathbf{W}, \mathbf{Z}$ s.t. $\mathbf{X} \approx \mathbf{W}\mathbf{Z}^\top$
$\min_{\mathbf{W},\mathbf{Z}} \mathcal{L}(\mathbf{W}, \mathbf{Z}) :=$
$\frac{1}{2} \sum_{(d,n)\in\Omega} f_{dn} \left[x_{dn} - \left(\mathbf{W}\mathbf{Z}^\top\right) dn\right]^2$
where $\mathbf{W} \in \mathbb{R}^{D\times K}$, $\mathbf{Z} \in \mathbb{R}^{N\times K}$, $K \ll D, N$, $\Omega \subseteq [D] \times [N]$ indices of non-zeros of the count matrix $\mathbf{X}$, f_{dn} are weights to each entry.

GloVe

A variant of word2vec.
$f_{dn} := \min\{1, (n_{dn}/n_{\max})^\alpha\}, \quad \alpha \in [0; 1]$ (e.g. $\alpha = \frac{3}{4}$)
Note: Choosing K; K e.g. 50, 100, 500

Training

- Stochastic Gradient Descent (SGD) ($\Theta(K)$ per step $\rightarrow$ easily parralelizable)
- Alternating Least-Squares (ALS)

Skip-Gram Model

- Uses binary classification (logistic regression objective), to separate real word pairs (w_d, w_n) from fake word pairs. Same inner product score = matrix factorization.
- Given w_d, a context word w_n is:
real = appearing together in a context window of size 5
fake = any word $w_{n'}$ sampled randomly: Negative sampling (also: Noise Contrastive Estimation)

Learning Representations of Sentences & Documents

- Supervised: For a supervised task (e.g. predicting the emotion of a tweet), we can use matrix factorization or CNNs. - Unsupervised:
Adding or averaging (fixed, given) word vectors,
Training word vectors such that adding/averaging works well
Direct unsupervised training for sentences (appearing together with context sentences) instead of words

Fast Text

Matrix factorization to learn document/sentence representations (supervised).
Given a sentence $s_n = (w_1, w_2, \ldots, w_m)$, let $\mathbf{x}_n \in \mathbb{R}^{|\mathcal{V}|}$ be the bag-of-words representation of the sentence.

$$\min_{\mathbf{W},\mathbf{Z}} \mathcal{L}(\mathbf{W}, \mathbf{Z}) := \sum_{s_n \text{ a sentence}} f\left(y_n \mathbf{W}\mathbf{Z}^\top \mathbf{x}_n\right)$$

where $\mathbf{W} \in \mathbb{R}^{1\times K}, \mathbf{Z} \in \mathbb{R}^{|\mathcal{V}|\times K}$ are the variables, and the vector $\mathbf{x}_n \in \mathbb{R}^{|\mathcal{V}|}$ represents our n-th training sentence.
Here f is a linear classifier loss function, and $y_n \in \{\pm 1\}$ is the classification label for sentence $\mathbf{x}_n$.

Language Models

Selfsupervised training:

- Can a model generate text? train classifier to predict the continuation (next word) of given text
- Multi-class: Use soft-max loss function with a large number of classes D = vocabulary size
- Binary classification: Predict if next word is real or fake (i.e. as in word2vec)
- Impressive recent progress using large models, such as transformers

---