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 outliars.
- 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 outliars.

- 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 $\forall\ \mathbf{u}, \mathbf{v} \in \mathbb{R}^D, 0 \le \lambda \le 1$:

$h(\lambda\mathbf{u} + (1-\lambda)\mathbf{v}) \le \lambda h(\mathbf{u}) + (1-\lambda)h(\mathbf{v})$

Stirctly 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:

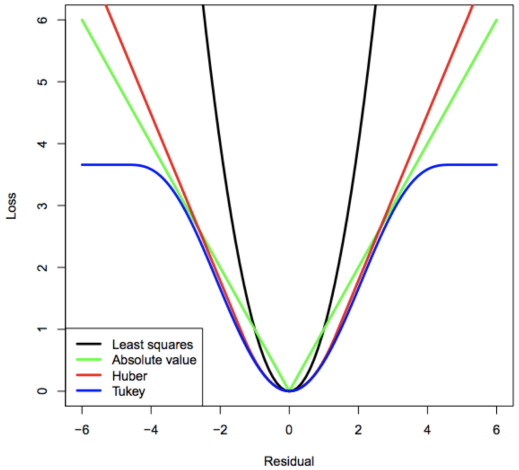
$$\mathcal{H}uber(e) := \begin{cases} \frac{1}{2}e^2 & ,\text{if } |e| \le \delta \\ \delta|e| - \frac{1}{2}\delta^2 & ,\text{if } |e| > \delta \end{cases} \quad \text{convex,}$$

differentiable, and robust to outliers but setting δ is not easy.

- Tukey's bisquare loss:

$$\frac{\partial\mathcal{L}}{\partial e} := \begin{cases} e\{1 - e^2/\delta^2\}^2 & ,\text{if } |e| \le \delta \\ 0 & ,\text{if } |e| > \delta \end{cases} \quad \text{non-convex,}$$

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:

$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$, 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

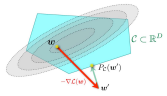
$\theta\mathbf{u} + (1-\theta)\mathbf{v} \in \mathcal{C}$

- 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:

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

2nd order: $\mathcal{L}$ 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 $\mathbf{w}, \mathbf{w}'$

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

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

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,

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

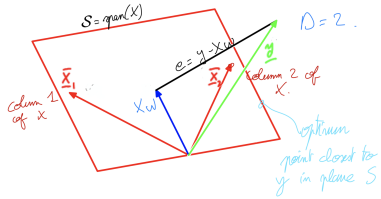
$\mathbf{S}$ diagonal matrix with non-negative entries, $\mathbf{V}^\top\mathbf{v}$ vector.

- 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: To see this assume first that $\operatorname{rank}(\mathbf{X}) < D$. 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 $\operatorname{rank}(\mathbf{X}^\top\mathbf{X}) < D$. 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

Unfortunately, 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.

Can we solve least squares if $\mathbf{X}$ is rank deficient? Yes, using a linear system solver.

Closed-form solution for MAE

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

Overfitting

Models can be too limited or they can be too rich. In the first case we cannot find a function that is a good fit for the data in our model. We then say that we underfit. In the second case we have such a rich model family that we do not just fit the underlying function but we in fact fit the noise in the data as well. We then talk about an overfit. Both of these phenomena are undesirable. This discussion is made more difficult since all we have is data and so we do not know a priori what part is the underlying signal and what part is noise.

Underfitting with Linear Models

It is easy to see that linear models might underfit. Consider a scalar case as shown in the figure below.

The solid curve is the underlying function and the circles are the actual data. E.g., we assume that there is a scalar function $g(x)$ but that we do not observe $g(x_n)$ directly but only a noisy version of it, $y_n = g(x_n) + Z_n$, where Z_n is the noise. The noise might be due for example to some measurement inaccuracies. The y_n are shown as blue cirles. If our model family consists of only linear functions of the scalar input x, i.e., $\mathcal{F} = \{f_w(x) = wx\}$, where w is a scalar constant (the slope of the function), then it is clear that we

cannot match the given function accurately, regardless how many samples we get and how small the noise is. We therefore will underfit.

Extended/Augmented Feature Vectors From the above example it might seem that linear models are too simple to ever overfit. But in fact, linear models are highly prone to overfitting, much more so than complicated models like neural nets.

Since linear models are inherently not very rich the following is a standard "trick" to make them more powerful.

In order to increase the representational power of linear models we typically "augment" the input. E.g., if the input (feature) is one-dimensional we might add a polynomial basis (of arbitrary degree M),

$$\boldsymbol{\phi}(x_n) := \left[1, x_n, x_n^2, x_n^3, \ldots, x_n^M\right]$$

so that we end up with an extended feature vector. We then fit a linear model to this extended feature vector $\phi(x_n)$:

$$y_n \approx w_0 + w_1 x_n + w_2 x_n^2 + \ldots + w_M x_n^M =: \boldsymbol{\phi}(x_n)^\top \mathbf{w}$$

Overfitting with Linear Models

In the following four figures, circles are data points, the green line represents the "true function", and the red line is the model. The parameter M is the maximum degree in the polynomial basis.

For $M = 0$ (the model is a constant) the model is underfitting and the same is true for $M = 1$. For $M = 3$ the model fits the data fairly well and is not yet so rich as to fit in addition the small "wiggles" caused by the noise. But for $M = 9$ we now have such a rich model that it can fit every single data point and we see severe overfitting taking place. What can we do to avoid overfitting? If you increase the amount of data (increase N, but keep M fixed), overfitting might reduce. This is shown in the following two figures where we again consider the same model complexity $M = 9$ but we have extra data ($N = 15$ or even $N = 100$).

A Word About Notation

If it is important to distinguish the original input $\mathbf{x}$ from the augmented input then we will use $\phi(\mathbf{x})$ to denote this augmented input vector. But we can consider this augmentation as part of the pre-processing, and then we might simply write $\mathbf{x}$ to denote the input. This will save us a lot of notation.

Additional Materials

Read about overfitting in the paper by Pedro Domingos (Sections 3 and 5 of "A few useful things to know about machine learning").

Maximum Likelihood

In the previous lecture 3a we arrived at the least-squares problem in the following way: we postulated a particular cost function (square loss) and then, given data, found that model that minimizes this cost function. In the current lecture we will take an alternative route. The final answer will be the same, but our starting point will be probabilistic. In this way we find a second interpretation of the least-squares problem.

Gaussian distribution and independence

Recall the definition of a Gaussian random variable in $\mathbb{R}$ with mean μ and variance σ^2. It has a density of

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

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

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

Also recall that two random variables X and Y are called independent when $p(x, y) = p(x)p(y)$.

A probabilistic model for least-squares

We assume that our data is generated by the model,

$$y_n = \mathbf{x}_n^\top \mathbf{w} + \epsilon_n$$

where the ϵ_n (the noise) is a zeromean Gaussian random variable with variance σ^2 and the noise that is added to the various samples is independent of each other, and independent of the input. Note that the model $\mathbf{w}$ is unknown.

Therefore, given N samples, the likelihood of the data vector $\mathbf{y} = (y_1, \cdots, y_N)$ given the input $\mathbf{X}$ (each row is one input) 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

Instead of maximizing the likelihood, we can take the logarithm of the likelihood and maximize it instead. Expression is called the loglikelihood (LL).

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

Compare the LL to the MSE (mean squared error)

$$\mathcal{L}_{\mathrm{LL}}(\mathbf{w}) = -\frac{1}{2\sigma^2}\sum_{n=1}^N \left(y_n - \mathbf{x}_n^\top \mathbf{w}\right)^2 + \text{cnst}$$

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

Maximum-likelihood estimator (MLE)

It is clear that maximizing the LL is equivalent to minimizing the MSE:

$\arg\min_{\mathbf{w}} \mathcal{L}_{\mathrm{MSE}}(\mathbf{w}) = \arg\max_{\mathbf{w}} \mathcal{L}_{\mathrm{LL}}(\mathbf{w})$.

This gives us another way to design cost functions.

MLE can also be interpreted as finding the model under which the observed data is most likely to have been generated from (probabilistically). This interpretation has some advantages that we discuss now.

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)

$\mathbf{w}_{\mathrm{MLE}} \longrightarrow^p \mathbf{w}_{\mathrm{true}}$ in probability

The MLE is asymptotically normal, i.e.,

$(\mathbf{w}_{\mathrm{MLE}} - \mathbf{w}_{\mathrm{true}}) \longrightarrow^d \frac{1}{\sqrt{N}}\mathcal{N}\left(\mathbf{w}_{\mathrm{MLE}} \mid \mathbf{0}, \mathbf{F}^{-1}(\mathbf{w}_{\mathrm{true}})\right)$

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 $(\mathbf{w}_{\mathrm{MLE}}) = \mathbf{F}^{-1}(\mathbf{w}_{\mathrm{true}})$

Another example

We can replace Gaussian distribution by a Laplace distribution.

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

Regularization

We have seen that by augmenting the feature vector we can make linear models as powerful as we want. Unfortunately this leads to the problem of overfitting. Regularization is a way to mitigate this undesirable behavior.

We will discuss regularization in the context of linear models, but the same principle applies also to more complex models such as neural nets.

Regularization

Through regularization, we can penalize complex models and favor simpler ones:

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

The second term Ω is a regularizer, measuring the complexity of the model given by $\mathbf{w}$.

L_2-Regularization: Ridge Regression

The most frequently used regularizer is the standard Euclidean norm (L_{2-} norm), that is

$$\Omega(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2$$

where $\|\mathbf{w}\|_2^2 = \sum_i w_i^2$. Here the main effect is that large model weights w_i will be penalized (avoided), since we consider them "unlikely", while small ones are ok. 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: Differentiating and setting to zero:

$$\mathbf{w}^\star_{\mathrm{ridge}} = \left(\mathbf{X}^\top \mathbf{X} + \lambda' \mathbf{I}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

(here for simpler notation $\frac{\lambda'}{2N} = \lambda$)

Ridge Regression to Fight III-Conditioning

The eigenvalues of $\left(\mathbf{X}^\top \mathbf{X} + \lambda' \mathbf{I}\right)$ are all at least λ' and so the inverse always exists. This is also referred to as lifting the eigenvalues.

Proof: Write the Eigenvalue decomposition of $\mathbf{X}^\top \mathbf{X}$ as $\mathbf{U}\mathbf{S}\mathbf{U}^\top$. We then have

$$\mathbf{X}^\top \mathbf{X} + \lambda' \mathbf{I} = \mathbf{U}\mathbf{S}\mathbf{U}^\top + \lambda' \mathbf{U}\mathbf{I}\mathbf{U}^\top$$
$$= \mathbf{U}\left[\mathbf{S} + \lambda' \mathbf{I}\right]\mathbf{U}^\top$$

We see now that every Eigenvalue is "lifted" by an amount λ'.

Here is an 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

$$\frac{\mathbf{v}^\top \left(\mathbf{X}^\top\mathbf{X} + \lambda'\mathbf{I}\right)\mathbf{v}}{\mathbf{v}^\top\mathbf{v}} \geq \frac{\lambda'\mathbf{v}^\top\mathbf{v}}{\mathbf{v}^\top\mathbf{v}} = \lambda'$$

L_1-Regularization: The Lasso

As an alternative measure of the complexity of the model, we can use a different norm. A very important case is the L_1-norm, leading to L_1- regularization. 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

$$\|\mathbf{w}\|_1 := \sum_i |w_i| .$$

The figure above shows a "ball" of constant L_1 norm. To keep things simple 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. We claim that for the 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.
Why is this the case? Assume that a genie tells you the L_1 norm of the optimum solution. Draw the L_1-ball with that norm value (think of 2D to visualize it). So now you know that the optimal point is somewhere on the surface of this "ball". Further you know that there are ellipsoids, all with the same mean and rotation that describes the equal error surfaces incurred by the first term. The optimum solution is where the "smallest" of these ellipsoids just touches the L_1-ball. Due to the geometry of this ball this point is more likely to be on one of the "corner" points. In turn, sparsity is desirable, since it leads to a "simple" model.
How do we see the claim that (1) describes and ellipsoid? 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 $\mathbf{y} = \mathbf{y}_\parallel + \mathbf{y}_\perp$, where $\mathbf{y}_\parallel$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

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

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-section.

Additional Notes

Other Types of Regularization

Popular methods such as shrinkage, dropout and weight decay (in the context of neural networks), early stopping of the optimization are all different forms of regularization.
Another view of regularization: The ridge regression formulation we have seen above is similar to the following constrained problem (for some $\tau > 0$).

$$\min_{\mathbf{w}} \frac{1}{2N}\sum_{n=1}^{N}\left(y_n - \mathbf{x}_n^\top\mathbf{w}\right)^2, \quad \text{such that } \|\mathbf{w}\|_2^2 \leq \tau$$

The following picture illustrates this.
Figure 1: Geometric interpretation of Ridge Regression. Blue lines indicating the level sets of the MSE cost function.
For the case of using L_1 regularization (known as the Lasso, when used with MSE) we analogously consider

$$\min_{\mathbf{w}} \frac{1}{2N}\sum_{n=1}^{N}\left(y_n - \mathbf{x}_n^\top\mathbf{w}\right)^2, \quad \text{such that } \|\mathbf{w}\|_1 \leq \tau$$

This forces some of the elements of $\mathbf{w}$ to be strictly 0 and therefore enforces sparsity in the model (some features will not be used since their coefficients are zero).

- Why does L_1 regularizer enforce sparsity? Hint: Draw the picture similar to above, and locate the optimal solution.
- Why is it good to have sparsity in the model? Is it going to be better than least-squares? When and why?

Ridge Regression as MAP estimator

Recall that classic least-squares linear regression can be interpreted as the maximum likelihood estimator:

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

In step (a) on the right we wrote down the negative of the log of the likelihood. The maximum likelihood criterion choses that parameter $\mathbf{w}$ that minimizes this quantity (i.e., maximizes the likelihood). In step (b) we factored the likelihood. The usual assumption is that the choice of the input samples $\mathbf{x}_n$ does not depend on the model parameter (which only influces the output given the input. Hence, in step (c) we removed the conditioning. Since the factor $p(\mathbf{X})$ does not depend on $\mathbf{w}$, i.e., is a constant wrt to w) we can remove it. This is done in step (d). In step (e) we used the assumption that the samples are iid. In step (f) we then used our assumption that the samples have the form $y_n = \mathbf{w}_n^\top\mathbf{w} + Z_n$, where Z_n is a Gaussian noise with mean zero and variance σ_2. The rest is calculus.
Ridge regression has a very similar interpretation. Now 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 $\frac{1}{\lambda}$.

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

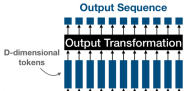
In step (a) we used Bayes' law. In step (b) and (c) we eliminated quantities that do not depend on $\mathbf{w}$.

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

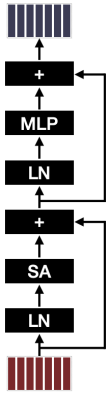


- Multi-Layer Perceptron (MLP): mixes information within each token

- Skip connections are widely used

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

Text Token Embeddings



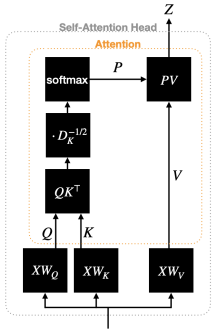
- Tokenization: split the input text into a sequence of input tokens (typically word fragments + some special symbols) according to some predefined tokenizer procedure:
- Convert each token ID $i \in \{1, ..., N_{vocab}\}$ into a real-valued vector $\mathbf{w}_i \in \mathbb{R}^D$
- This can be seen as a matrix multiplication $\mathbf{W} \cdot \mathbf{e}_i = \mathbf{W}_{:,i} = \mathbf{w}_i$ (with $\mathbf{W} \in \mathbb{R}^{\mathrm{D} \times \mathrm{N}_{\mathrm{vocab}}}$)
- $\mathbf{W}$ 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

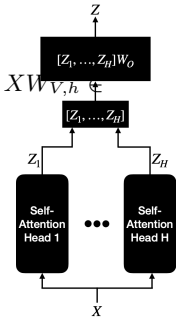
- V, K, Q are all derived from the same input token sequence $X \in \mathbb{R}^{T \times D}$
- Values : $V = XW_V \in \mathbb{R}^{T \times D}$, $W_V \in \mathbb{R}^{D \times D}$
- Keys : $K = XW_K \in \mathbb{R}^{T \times D_K}$, $W_K \in \mathbb{R}^{D \times D_K}$
- Queries : $Q = XW_Q \in \mathbb{R}^{T \times D_K}$, $W_Q \in \mathbb{R}^{D \times D_K}$
- 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

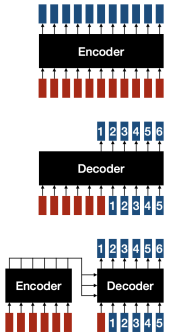
Output Transformations

- 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



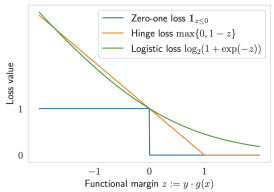
- Encoders (e.g., classification): They produce a fixed output size and process all inputs simultaneously
- Decoders (e.g., ChatGPT): Auto-regressively sample the next token as $x_{t+1} \sim sofimax(f(x_1, ..., x_t))$ and use it as new input token. Capable of generating responses of arbitrary length.
- 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: $R_\varepsilon(f) = \mathbb{E}_{\mathcal{D}}\left[\max_{\hat{x}, \|\hat{x} - X\| \leq \varepsilon} 1_{f(\hat{x}) \neq Y}\right]$

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) $\|\hat{x} - x\| \leq \varepsilon$ b) the model f makes a mistake on it.
- Trivial case: x already missclassified → no action required
- General case: find $\hat{x}$ such that at$f(\hat{x}) \neq y$ and $\|\hat{x} - x\| \leq \varepsilon$ 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 $\max_{\hat{x}, \|\hat{x} - X\| \leq \varepsilon} 1_{f(\hat{x}) \neq Y} \to \max_{\hat{x}, \|\hat{x} - X\| \leq \varepsilon} \ell(yg(\hat{x}))$
- decreasing, margin-based (i.e., dependent on $y * g(x)$) classification losses



White-Box attacks

- Solve $\max_{\hat{x}, \|\hat{x} - X\| \leq \varepsilon} \ell(yg(\hat{x}))$ knowing g
- $\nabla_x \ell(yg(x)) = y\ell'(yg(x)) \nabla_x g(x)$, with $y\ell'(yg(x)) \leq 0$ 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 $\tilde{\ell}(x) := \ell(yg(x))$

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

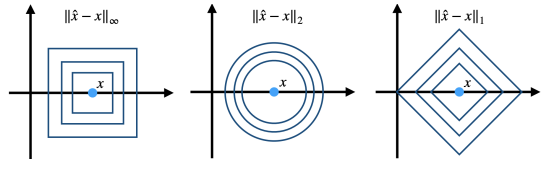
- We need to maximize the inner product under a norm constraint, i.e. find the optimal local update
- This is a simple problem for which we can get a closed-form solution depending on the norm used to measure the perturbation size $||\delta||$

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:
$\delta^{t+1} = \Pi_{B_2(\epsilon)}[\delta^t + \alpha \cdot \frac{\nabla \tilde{\ell}(x + \delta^t)}{\|\nabla \tilde{\ell}(x + \delta^t)\|_2}]$
$\Pi_{B_2(\varepsilon)}(\delta) = \begin{cases} \varepsilon \cdot \delta / \|\delta\|_2, & \text{if } \|\delta\|_2 \geq \varepsilon \\ \delta \text{ otherwise} \end{cases}$
- ℓ_∞ norm:
$\delta^{t+1} = \Pi_{B_\infty(\varepsilon)}\left[\delta^t + \alpha \cdot \text{sign}(\nabla \tilde{\ell}(x + \delta^t))\right]$,
$\Pi_{B_\infty(\varepsilon)}(\delta)_i = \begin{cases} \varepsilon \cdot \text{sign}(\delta_i), & \text{if } |\delta_i| \geq \varepsilon \\ \delta_i \text{ otherwise} \end{cases}$
- 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:
$\min_\theta R_\varepsilon(f_\theta) = \mathbb{E}_{\mathcal{D}}\left[\max_{\hat{x}, \|\hat{x}-X\| \leq \varepsilon} 1_{f(\hat{x}) \neq Y}\right]$
- $\mathcal{D}$ unknown $\rightarrow$ approximate it with a sample average + classification loss is non-continuous $\rightarrow$ use a smooth loss $\Rightarrow$
$\min_\theta \frac{1}{N}\sum_{n=1}^{N} \max_{\hat{x}_n, \|x_n - \hat{x}_n\| \leq \varepsilon} \ell(y_n g_\theta(\hat{x}_n))$
1) $\forall x_n$, $\hat{x}_n^\star \approx \arg\max_{||x_n - \hat{x}_n|| \leq \varepsilon} \ell(y_n g_\theta(\hat{x}_n))$
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: $\arg\max_{\hat{y} \in \{\pm 1\}} p(\hat{y} \mid x_1) =$
$\arg\max_{\hat{y} \in \{\pm 1\}} \frac{p(x_1 \mid \hat{y}) p(\hat{y})}{p(x_1)} = \arg\max_{\hat{y} \in \{\pm 1\}} p(x_1 \mid \hat{y})$
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 $x_i = y a_i + Z_i$
$\arg\max_{\hat{y} \in \{\pm 1\}} p(\hat{y} \mid x)$
$= \arg\max_{\hat{y} \in \{\pm 1\}} \prod_{i=1}^{d} p(x_i \mid \hat{y})$
$= \arg\max_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^{d} \log p(x_i \mid \hat{y})$
$= \arg\max_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^{d} \log \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x_i - \hat{y} a_i)^2}$
$= \arg\min_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^{d} (x_i - \hat{y} a_i)^2$
$= \arg\min_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^{d} (x_i^2 - 2 x_i \hat{y} a_i + \hat{y}^2 a_i^2)$
$= \arg\max_{\hat{y} \in \{\pm 1\}} \hat{y} \sum_{i=1}^{d} x_i a_i$
$\hat{y}\sum_{i=1}^{d} x_i a_i = \hat{y} y (\sum_{i=1}^{d} a_i^2) + \hat{y}\sum_{i=1}^{d} a_i Z_i = \hat{y} y (1 + \log(d)) + \hat{y} Z$ where $Z := \sum_{i=1}^{d} a_i Z_i \sim \mathcal{N}(0, 1 + \log d)$
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.

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

$X \approx W Z^T$, $W \in \mathbb{R}^{D \times K}$, $Z \in \mathbb{R}^{N \times K}$ tall matrices $K << N, D$
$\min_{W,Z} \ \mathcal{L}(W, Z) := \frac{1}{2}\sum_{(d,n) \in \mathbb{Q}} [x_{dn} - (W Z^T)_{dn}]^2$
- 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

$\frac{1}{2}\sum_{(d,n) \in \Omega} [x_{dn} - (W Z^T)_{dn}]^2 + \frac{\lambda_w}{2} ||W||_{\text{Frob}}^2 + \frac{\lambda_z}{2} ||Z||_{\text{Frob}}^2 \ \lambda_w, \lambda_z \in \mathbb{R} > 0$

Stochastic Gradient Descent

$\mathcal{L} = \frac{1}{|\Omega|}\sum_{(d,n) \in \Omega} \underbrace{\frac{1}{2}[x_{dn} - (\mathbf{W}\mathbf{Z}^\top)_{dn}]^2}_{f_{d,n}}$

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:
$\frac{1}{2}\sum_{d=1}^{D}\sum_{n=1}^{N} \left[x_{dn} - \left(W\mathbf{Z}^\top\right)_{dn}\right]^2$
$= \frac{1}{2}\|\mathbf{X} - \mathbf{W}\mathbf{Z}^\top\|_{Frob}^2$
- We first minimize w.r.t. Z for fixed W and then minimize W given Z (closed form solutions):
$Z^\top := (\mathsf{W}^\top\mathsf{W} + \lambda_z \mathsf{I}_K)^{-1}\mathsf{W}^\top\mathsf{X}$
$\mathbf{W}^\top := (\mathbf{Z}^\top\mathbf{Z} + \lambda_w \mathbf{I}_K)^{-1}\mathbf{Z}^\top\mathbf{X}^\top$
- 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. $\frac{1}{2}\sum_{(d,n) \in \Omega} \left[x_{dn} - \left(\mathsf{W}\mathsf{Z}^\top\right)_{dn}\right]^2$
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

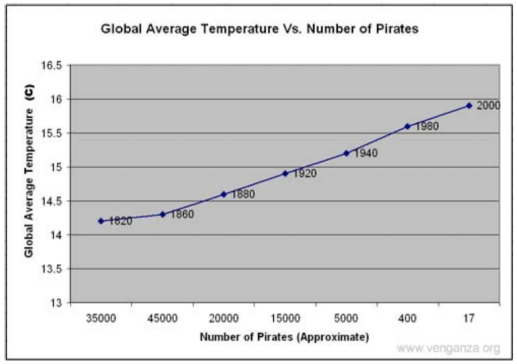
Selfsupervised training:

- 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

1 This is RGB red text.

For $x \in [r_{i-1}, r_i]$
$r(x) = \tilde{a}_1 x + \tilde{b}_1 + \sum_{j=2}^{m} \tilde{a}_j (x - \tilde{b}_j)_+$

For $x \in [r_{i-1}, r_i]$
$r(x) = \tilde{a}_1 x + \tilde{b}_1 + \sum_{j=2}^{m} \tilde{a}_j (x - \tilde{b}_j)_+$



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

. . . . . .

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.
