CS335/337 : AI & ML

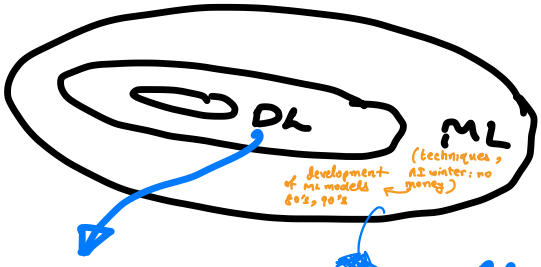
(Prof. Preeti Jyothi)

learn: Make accurate predictions or decisions based on data by optimizing model

models are lossy by nature.

"All models are wrong but some are useful"

Performance measure to identify which f^ is better



AI (1950s: perceptrons) ← Origin: Perceptrons (lin. models)

NNs, deep learning models

SVMs, MCMC, hidden markov, CNNs

ML & Statistics

ML ⊆ CS → scalability, heuristics, efficient algorithms, finite sample

Stats ⊆ math → proving asymptotic bounds

When ML?

1. For tasks that are easy for humans but are complex for computer systems to emulate (hard to code)
   - Computer vision (solved problem)
   - Natural language (translation, Q/A, identifying sentiment)
   - Speech (recognise speech, speaking sentences naturally)
   - Game playing (play games like chess, Go, DOTA, Poker, etc.) {RL}

- Robotics (walking, jumping, displaying emotions) {usually not ML}
- Self driving, navigating maze, etc.

2. For tasks that are beyond human capabilities
- Analysis of large and complex datasets
- IBM Watson's Jeopardy machine (parse nat. lang.), health analyst (symptoms, history → medical diagnosis)

Kaggle: Community to solve various problems.
↳ Survey '17 – '21: Logistic reg > decision trees > Grad boosting > CNNs > Bayesian > Dense NNs > RNNs
models to be covered in this course.

1) Supervised learning: Decision trees, neural networks, etc.

Let $\mathcal{D}$ be a dataset with n training examples,

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

(or) $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in \mathcal{X}$,

$\mathcal{X}$ is feature space. ($x_i \in \mathbb{R}^d$, x_i is a d-dimensional input real-valued vector)

$y_i \in \mathcal{Y}$, $\mathcal{Y}$ is the label space

1. $\mathcal{Y} = \{0, 1\}$ or $\{-1, 1\}$, $|\mathcal{Y}| = 2$: Binary classification
2. $\mathcal{Y} = \{1, 2, \ldots, k\}$, $k > 2$: Multiclass classification
3. $\mathcal{Y} = \mathbb{R}$: Regression

Objective: Let the instances in $\mathcal{D}$ be drawn from some unknown (true) probability distribution P over $\mathcal{X} \times \mathcal{Y}$.

Find an $h : X \to Y$ s.t. for a new test instance (x, y), $h(x) = y$ with high probability ($h(x) \approx y$)

2) **Unsupervised learning**: k-means, PCA, mixture models

$D = \{x_1, \ldots, x_n\}$

Identify structural properties to identify clusters, partition the data D

3) **Reinforcement learning**: Not covered in course (CS747, CS748)

$h \in \mathcal{H}$ ($\mathcal{H}$: hypothesis class)

↓ choice of model (e.g. linear, etc.)

what is the best model

Course logistics : refer moodle slides

Course project : 3-4 members

↳ Pick doable problems, many datasets, many libraries

Lecture 2

(3/8/2023)

Preliminaries $\mathcal{X} \equiv$ Input / Feature / Attribute space

$\mathcal{Y} \equiv$ Output / Label / Response

Linear Regression : $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$

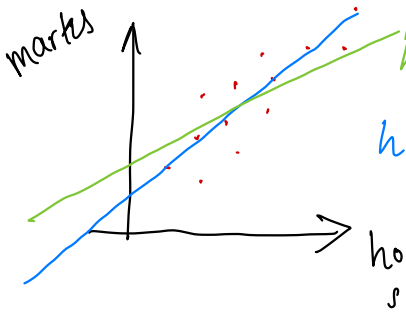
Consider a function $f : \mathcal{X} \to \mathcal{Y}$. Given a dataset D which consists of n-tuples (x, y) {training dataset} goal is to find a hypothesis h that ideally closely approximates f where, $h \in \mathcal{H}$ ($\mathcal{H}$ is hypothesis class)

Three important questions

1. What predictors are permissible (characterization of h & $\mathcal{H}$)? (Hypothesis class)
2. How do I measure if a predictor is good ? (loss/error function)
3. How do we find the best predictor ?

(Optimization problem)

1. Possible predictors



$$h_w(x) = w_0 + w_1 x, \quad w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ x \end{bmatrix}$$

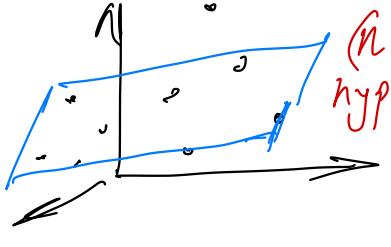
Vector notation $h_w(x) = w^T x$

($h_w(x)$ is a linear combination of features)

even if $x = \begin{bmatrix} 1 \\ x^2 \end{bmatrix}$, model is still linear regression

$$h_w(x) = w^T x, \quad w = \begin{bmatrix} w_1 \\ \dots \\ w_{d+1} \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ x_1 \\ \dots \\ x_d \end{bmatrix}$$

$$\mathcal{H} = \{h_w : w \in \mathbb{R}^{d+1}\}$$ (hypothesis class)



2. Quantify a measure of prediction

Loss function (error function) (objective)

$L(h, D)$

h: hypothesis, D: dataset

In case of linear regression, we can work with $L(w, D)$

Linear Regression

$$L(w, D_{test}) = (y - h_w(x))^2$$

least squares loss

$D_{test} = \{x, y\}$ (test instance)

$|y - \hat{y}|$ referred to as residuals

For absolute loss,

1. For outliers, it's more forgiving since grows slower than square loss
2. Analytic optimum issues due to non-diff.

$$L(w, D_{train}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

(omitted at times for simplicity)

$D_{train} = \{(x_i, y_i)\}_{i=1}^{n}$

Now, we stack x_i s to make

$$X = \begin{bmatrix} x_1^T \\ \dots \\ x_n^T \end{bmatrix}_{n \times (d+1)}, \quad Y = \begin{bmatrix} y_1 \\ \dots \\ y_n \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \\ \dots \\ w_d \end{bmatrix}_{(d+1) \times 1}$$

$$L(w, D_{train}) = \frac{1}{n} \|Y - Xw\|_2^2 \quad \text{(l2 norm squared)}$$

3. Finding the best predictor

$$h^* = \arg\min_h \; L(h, D_{train})$$

(training loss)

$$w^* = \arg\min_w \; L(w, D_{train})$$

$$w^*_{LS} = \arg\min_w \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

Least squares solution

Closed form solution for 3-dimensional data

$$w^* = \arg\min_w \underbrace{\sum_i (y_i - w^T x_i)^2}_{L_w}$$

Take the derivative of L_w w.r.t w, set to 0 and solve for w

$$\nabla_w L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \end{bmatrix} = 2\left(\sum_i (y_i - w^T x_i)\; x_i\right) = 0$$

$$\Rightarrow \sum_i y_i x_i - \sum_i (w^T x_i) x_i = 0$$

$$\Rightarrow w = (X^T X)^{-1} X^T Y$$

For data D, $h(x) = w_0 + w_1 x$

confirm that $w_1^* = \dfrac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$

$\bar{x} = \dfrac{\sum_i x_i}{n}$,

$\bar{y} = \dfrac{\sum_i y_i}{n}$

Completing the linear regression proof,

$\sum_i y_i x_i - \sum_i (w^T x_i) x_i = 0$

$\Rightarrow X^T Y - X^T X w = 0 \quad X^T = [x_1 \cdots x_n]$

$\Rightarrow w = (X^T X)^{-1} X^T y \quad Y = \begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix}$

Note: If $X^T X$ is not inv, we get multiple sol^{n}s e.g. infinite planes pass through a line, happens when cols of X are l.D. i.e. are "collinear" or $(n-2)$ hyperplane lying.

Lecture 3

(8/18/23)

1. Converting non-linear regression to linear regression

In the general regression case, h need not be linear. Consider the following simple case: $h_w = w_0 + w_1 x + w_2 x^2$

$\phi: \mathbb{R} \to \mathbb{R}^3$ by $\phi(x) = \begin{pmatrix} 1 \\ x \\ x^2 \end{pmatrix}$, $h_w(x) = w^T \phi(x)$

$$w = (\phi^T \phi)^{-1} \phi^T y, \text{ where } \phi = \begin{pmatrix} \phi(x_1)^T \\ \phi(x_2)^T \\ \cdots \\ \phi(x_n)^T \end{pmatrix}$$

(General case)

$\phi: \mathbb{R}^d \to \mathbb{R}^D$. $\phi(x) = \begin{pmatrix} \phi_1(x) \\ \cdots \\ \phi_D(x) \end{pmatrix}$ $(x_i \in \mathbb{R}^d)$

$h_w(x) = w^T \phi(x)$.

$$h(w, D) = \frac{1}{n} \sum_{i=1}^{n} (h_w(x_i) - y_i)^2 = \frac{1}{n} \sum_{i=1}^{n} (w^T \phi(x_i) - y_i)^2$$

So, we get normal solution,

$$w = (\phi^T \phi)^{-1} \phi^T y$$

where, $\phi = \begin{pmatrix} \phi(x_1)^T \\ \cdots \\ \phi(x_n)^T \end{pmatrix}$

Typical choices of basis transformations

1. Polynomial basis: $\phi(x) = (1, x, x^2, \cdots, x^{D-1})^T$ (the example we saw above)
2. Gaussian basis / Radial Basis functions: $\phi(x) = (1, e^{-\frac{|x-u_1|^2}{2\sigma^2}}, e^{-\frac{|x-u_2|^2}{2\sigma^2}}, \cdots, e^{-\frac{|x-u_D|^2}{2\sigma^2}})^T$
3. Fourier basis: $\phi(x) = (1, \cos(w_1^T x), \sin(w_1^T x), \cdots, \cos(w_D^T x), \sin(w_D^T x))^T$
4. Piecewise linear basis
5. Periodic basis

Some terminologies

- Training Data. The dataset $D = \{(x_i, y_i)\}_{i=1}^n$
- Training Error. $h(w, D) = \sum_{i=1}^{n} (h_w(x_i) - y_i)^2$
- Test error is a measure of the generalization of the model - better generalization is the holy grail of ML.
- Training - Dev - test $\approx$ 80 : 10 : 10
  (training the model) (tuning the hyper-parameters) (evaluate)
- Tuning Hyperparameters: Grid search, Random Search, Bayesian opt.

Underfitting and Overfitting

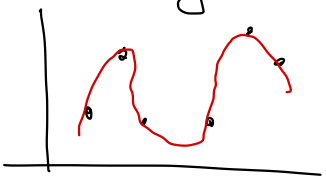
$\mathcal{H}$ = all lines parallel to the axes e.g. $h_c(x) = c$ or $x_i = c$

Too simple, cannot fit most data well – it is too simplistic and inflexible. Called "underfitting", where model is needed to be more complex.

Lecture 4

(08/08/25)

Overfitting



Training error : 0

Test error : high

How do we combat this?

1. Coarse strategy : tune k (order of polynomial) and evaluate on a dev set to pick a "good value".
2. Regularization : modify the loss function to explicitly constrain model complexity

$$h_{reg}(w, \mathcal{D}_{train}) = h_{MSE}(w, \mathcal{D}_{train}) + \lambda(R_w), \text{ where } \lambda \geq 0.$$

$R(w)$: Regulizer that penalizes or shrinks the weights in W

1st term: $h_{reg}(w, \mathcal{D}_{train})$ = measure of fit to training data

2nd term: $R(w)$ = measure of model complexity

Penalty of shrinkage-based regularizations

1. L_2 reg. : $R(w) = \|w\|_2^2$ ("shrinks" norm)
2. L_1 regularization: $R(w) = \|w\|_1$

] force w to not become arbitrarily large. more variance in coeff measured by $\|w\|$.

Induced regressions

1. Ridge regression (L_2-normalised regression)

$$w_{RIDGE} = \arg\min_w \|y - \phi w\|^2 + \lambda \|w\|_2^2$$

- Note : It is common to leave out w_0 (intercept) in ridge regression since, indep. of x_i, we might miss out on a best fit
- Note : we dropped the $\frac{1}{n}$ for $\|y - \phi w\|^2$ hence, λ must be scaled appropriately

Equivalent to

$$w_{RIDGE} = \arg\min_w \|y - \phi w\|^2 \quad \text{s.t. } \|w\|_2^2 \leq t$$

Solving,

$$\nabla L_{RIDGE}(w) = 0 \Rightarrow -2\phi^T(y - \phi w) + 2\lambda w = 0$$

$$w_{RIDGE} = (\phi^T\phi + \lambda I)^{-1}\phi^T y$$

($\phi^T\phi$: may not be invertible; λI: must be invertible! (if $\lambda > 0$))

{since $x^T\phi^T\phi x > 0 \ \forall x$, so $\phi^T\phi$ is p.s.d
so, $\phi^T\phi + \lambda I$, $\lambda > 0$, is p.d}

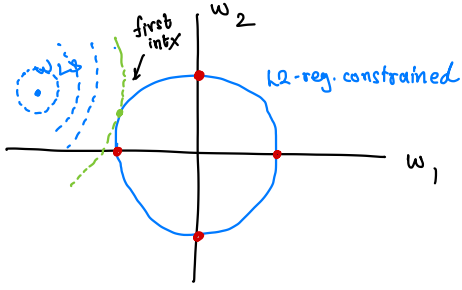
2. Lasso Regression (L_1-regularized regression)

$$w_{LASSO} = \arg\min_{w} \|y - \phi w\|_2^2 + \lambda \|w\|_1$$ (no closed form, but other ways to solve)

Solve for w_{LASSO} using

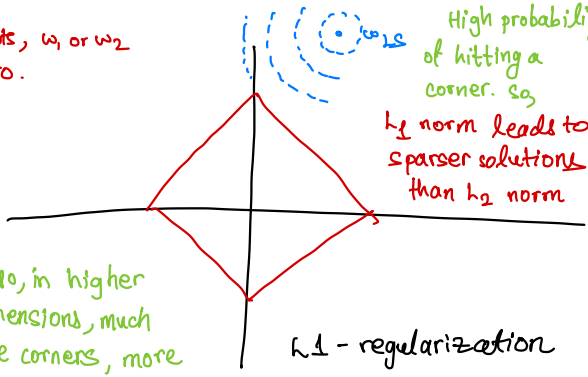
1. Quadratic Programming
2. Iterative optimization algorithm like "gradient descent"

L_1 regularization yields sparse weight vectors compared to L_2 regularization



L2-regularization

At red pts, w_1 or w_2 is zero.



High probability of hitting a corner. So, L_1 norm leads to sparser solutions than L_2 norm

Also, in higher dimensions, much more corners, more chance of hitting them

L1 - regularization

- When your data has outliers, sparsity is useful.

Lecture 5

(10/08/2023)

Gradient descent (GD)

GD is an iterative optimization algorithm

General template:

1. Initialize w (e.g. $w=0$)

Repeat
1. Choose a descent direction
2. Choose a step size
3. Update w

Until stopping criterion is met

Gradient descent

$\longleftrightarrow$ $w \leftarrow w_0$ (e.g. 0, etc.)

[for iteration $t = 0, 1, \ldots$]

$\longleftrightarrow$ Direction: $-\nabla L(w_t)$

$\longleftrightarrow$ $\alpha > 0$ (hyperparameter) (learning rate)

$\longleftrightarrow$ $w_{t+1} \leftarrow w_t - \alpha \nabla L(w_t)$

$\longleftrightarrow$ Common stopping criterion

$\|\nabla L(w_t)\|_2 \leq \varepsilon$, $\|w_{t+1} - w_t\| < \varepsilon$

ε is hyperparameter

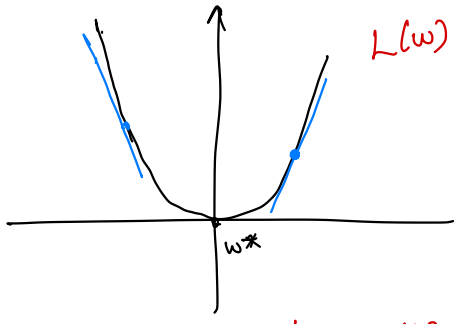
Gradient descent comes from

Gradient $\nabla L(w) = \begin{bmatrix} \frac{\partial L(w)}{\partial w_1} \\ \cdots \\ \frac{\partial L}{\partial w_n}(w) \end{bmatrix}$

gives you the direction of fastest increase.

descent update w in the direction of fastest decrease in L

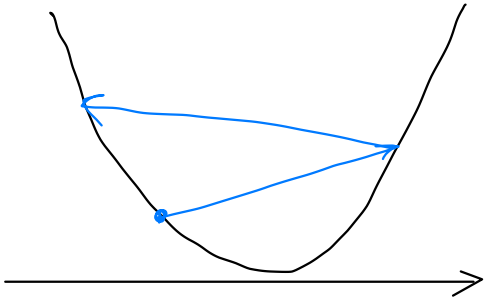
For 1D case and linear regression



What if step size is too small?
Takes longer to converge.

Step size determines how fast you converge.

What if step size is very large?



Loss can diverge in worst case, GD will not converge

What is the weight update rule in GD for linear regression?

$$w \leftarrow w - \alpha \nabla L(w)$$

$$L(w) = \sum_i (y_i - w^T x_i)^2$$

$$\nabla L(w) = -2 \sum_i (y_i - w^T x_i)\, x_i$$

$$= -2 \sum_i (y_i - \hat{y}_i)\, x_i$$

Substituting, we get

$$w \leftarrow w - \alpha \sum_i (\hat{y}_i - y_i)\, x_i$$

GD: $w \leftarrow w - \alpha \nabla L(w, D_{train})$

SGD (Stochastic Gradient descent)

$w \leftarrow w - \alpha \nabla L(w, D_{random})$

where, $D_{random} = \{(x_i, y_i)\}$, (x_i, y_i) is randomly sampled from D_{train}

- Issue: if example picked is outlier, not representative of D_{train}, we can take wrong steps, but we have convergence guarantees.

To counter SGD issues,

Mini-batch gradient descent:

$$w \leftarrow w - \alpha \nabla L(w, D_{batch})$$

where $D_{batch} = \{(x_i, y_i)\}_{i=1}^{B}$, B = batch size

Batch size dictated by RAM/GPU, basically, how many examples you can compute upon.

training becomes more stable with larger batch sizes, typically batch size = 32, 64.

(shuffle the dataset, then make one pass over all examples via mini batches, one pass over all examples is called an epoch)

Probabilistic view of linear Regression

For training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$. Let the target y_i have some noise defined as,

$$y_i = f(x_i) + \varepsilon_i, \quad \varepsilon_i \sim N(0, \sigma^2)$$

ε_i's are independent and identically distributed zero mean gaussians with same variance σ^2. ($Cov(\varepsilon_i, \varepsilon_j) = 0 \ \forall \ i \neq j$)

For linear regression,

$$\Leftrightarrow \quad y_i = w^T x_i + \varepsilon_i, \ \varepsilon_i \sim N(0, \sigma^2)$$

$$\Leftrightarrow \quad y_i \sim N(w^T x_i, \sigma^2)$$

$$\Leftrightarrow \quad P(y_i | x_i, w) = N(w^T x_i, \sigma^2)$$

↳ Likelihood

For training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$,

$$P(y_1, y_2 \cdots y_n | x_1 \cdots x_n, w) = \prod_i P(y_i | x_i, w)$ (y_i's are conditionally independent given x_i's)

$$\log P(y_1, y_2 \cdots y_n | x_1 \cdots x_n, w) = \prod_i \log P(y_i | x_i, w)$ (log likelihood)

The probabilities can get really close to zero, so high chance of numerical underflow, log probabilities are much more well behaved.

We want to find a w which maximises this likelihood, which will reduce to mean squared error.

14/08/2023

Lecture 6

$$Y = w^T X + \varepsilon, \varepsilon \sim N(0, \sigma^2)$$ (multivariate gaussian)

$$\Leftrightarrow \quad Y \sim N(w^T X, \sigma^2)$$ (examples stacked across columns)

For training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

$$P(y_1, y_2, \cdots, y_n | x_1, \cdots x_n) = \prod_i P(y_i | x_i, w)$$

(likelihood)

$$\log P(y_1, y_2, \cdots, y_n | x_1, \cdots x_n) = \prod_i \log P(y_i | x_i, w)$$

(log-likelihood)

Maximum likelihood estimation : Find parameters w that maximize the probability of the observed data.

$$w_{MLE} = \arg\max_w \sum_i \log P(y_i | x_i, w)$$

Motivating Example

You want to estimate the probability of a biased coin landing on heads (θ)
say your data is N coin tosses, with N_H heads and N_T tails (i.i.d).
What is the MLE estimate of θ?

$$P(D|\theta) = \theta^{N_H}(1-\theta)^{N_T}$$

$$\theta_{MLE} = \arg\max_\theta \log P(D|\theta) = \arg\max_\theta (N_H \log\theta + N_T \log(1-\theta))$$

i.e. $\frac{N_H}{\theta} + \frac{N_T}{1-\theta}(-1) = 0 \Rightarrow (1-\theta)N_H = \theta \cdot N_T$

$$\Rightarrow N_H = \theta\cdot(N_H + N_T) = \theta \cdot N$$

$$\Rightarrow \boxed{\theta = \frac{N_H}{N}}$$

Q Say you have a coin with bias $p \in \{0.4, 0.6\}$
Data; 3 coin tosses, 2H, 1T
What is the MLE of p.

Ans $P(D|p) \propto p^2(1-p)$

at $p=0.4$, $(0.4)^2(0.6)$ Hence, MLE $= 0.6$
$p=0.6$, $(0.6)^2(0.4)$

MLE for linear regression

$$w_{MLE} = \arg\max_w \sum_{i=1}^n \log P(y_i|x_i, w)$$

$$= \arg\max_w \sum_{i=1}^n \log \mathcal{N}(w^T x_i, \sigma^2)$$

$$= \arg\max_w \sum_{i=1}^n \log\left[\frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(y_i - w^T x_i)^2}{2\sigma^2}}\right]$$

$$= \arg\max_w \sum_{i=1}^n \left[C - \frac{(y_i - w^T x_i)^2}{2\sigma^2}\right]$$

$$w_{MLE} = \arg\min_w \sum_{i=1}^n (y_i - w^T x_i)^2 \quad \text{(least squares error solution)}$$

Homework: If we have a laplacian noise instead of gaussian, we get L_1 norm
i.e. least absolute error.

Note: The limitation of MLE is the tendency to overfit if you have limited data. (entirely data dependent).

Bayesian Parameter Estimation

In ML, observations are random variables, but parameters are not. In the bayesian framework, parameters are also random variables.
⇒ Underlying prior distribution which encodes beliefs about parameters before observing data.

Note: Prior is a pre-determined choice, parametrized by some vars, these parameters will be further hyperparameters.

Maximum Aposteriori Estimate (MAP)

$$P(\theta|D) \propto P(D|\theta)\,P(\theta)$$

posterior — likelihood — prior

$$\theta_{MAP} = \arg\max_\theta P(\theta|D) = \arg\max_\theta P(D|\theta)P(\theta) = \arg\max_\theta \log P(D|\theta) + \log P(\theta)$$

(MLE)

Coin example (MAP estimate)

$$P(D|\theta) = \theta^{N_H}(1-\theta)^{N_T}$$

what is a good prior on θ? **Beta distribution** is a good candidate for a prior

$$P(\theta,\alpha,\beta) = \frac{1}{C}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

We want to pick a prior so that posterior has a similar functional form as the prior

Conjugate Priors

For a likelihood $P(D|\theta)$ coming from a family of distributions d_1, a prior is said to be conjugate prior (from a family d_2) if the posterior distribution also comes from family d_2.

Coin example $B(N_H, N_T, \theta) = \theta^{N_H}(1-\theta)^{N_T}$

$\text{Beta}(\theta,\alpha,\beta) = C\,\theta^{\alpha-1}(1-\theta)^{\beta-1}$

$$P(\theta|D) \propto P(D|\theta)P(\theta)$$
$$\propto \theta^{N_H+\alpha-1}(1-\theta)^{N_T+\beta-1}$$
$$\propto \text{Beta}(\theta, N_H+\alpha, N_T+\beta)$$

Extra reading: Conjugate priors drawn from exponential families usually e.g. for bernoulli, binomial likelihood, beta distribution works. for multinomial, dirichlet prior for gaussian, gaussian prior

MLE ≡ unreg. linear regression
MAP ≡ reg. linear regression

Lecture 7

(17/08/2023)

Recap: MAP estimation

$$\theta_{MAP} = \arg\max_\theta \log P(D|\theta) + \log P(\theta) = \arg\max_\theta \log P(\theta|D)$$

Point estimates — Posterior probability distribution

Coin toss problem $B(n_H, n_T, \theta) = \theta^{n_H}(1-\theta)^{n_T}$

$\text{Beta}(\theta,\alpha,\beta) = \frac{1}{C}\theta^{\alpha-1}(1-\theta)^{\beta-1}$ [C is a normalisation constant]

$P(\theta|D) \propto \text{Beta}(\theta, n_H+\alpha, n_T+\beta)$ {beta distribution is a conjugate prior for the bernoulli/binomial likelihood}

$$\theta_{MAP} = \arg\max_\theta P(\theta|D) = \arg\max_\theta (n_H+\alpha-1)\log\theta + (n_T+\beta-1)\log(1-\theta)$$

$$\boxed{\theta_{MAP} = \frac{n_H+\alpha-1}{n+\alpha+\beta-2}}$$

$\frac{n_H+\alpha-1}{n+\alpha+\beta-2}$ ← (pseudocounts)

Note: $\theta_{MAP} \to \theta_{MLE}$ as $n \to \infty$ (Bernstein-von Mises theorem)

MAP estimate for Linear Regression

Consider the following Gaussian prior over W,

$$P(W) = \mathcal{N}\left(0, \frac{1}{\lambda} I\right) = \left(\frac{\lambda}{2\pi}\right)^{d/2} \exp\left(-\frac{\lambda}{2} W^T W\right)$$ ← follows

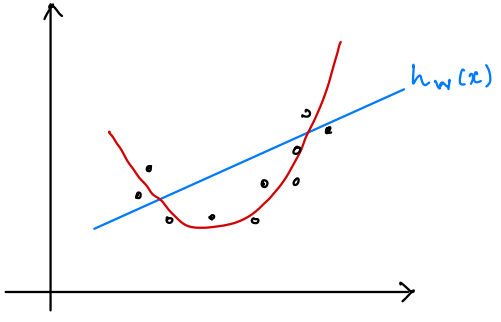
$$\left[\underset{\mu \in \mathbb{R}^d}{\mathcal{N}(\mu, \Sigma)} = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)\right]$$

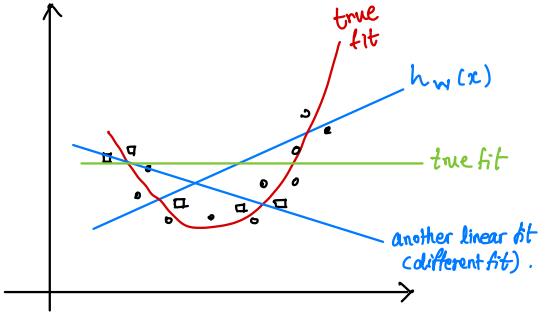
$$\begin{aligned} W_{MAP} &= \arg\max_W \log P(D|W) + \log P(W) \\ &= \arg\max_W \frac{-1}{2\sigma^2} \sum_i (y_i - W^T x_i)^2 - \frac{\lambda}{2} \|w\|_2^2 \\ &= \arg\min_W \frac{1}{2\sigma^2} \sum_i (y_i - W^T x_i)^2 + \frac{\lambda}{2} \|w\|_2^2 \end{aligned}$$ {ridge or L_2 regularised regression}

Note: If we use laplace prior instead of gaussian, we would get a L_1 regularisation.

(Laplacian) $f(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$

Bias and Variance of Estimators





1. How much does the mean of fit differ (linear, here) from true fit ← Bias
2. What is the variance across fits when training set varies ← Variance

] inherent to choice of hypothesis class.

3. Irrecoverable error (unavoidable)] inherent to data

Bias - Variance Analysis for Linear Regression

Let $y = f(x) + \varepsilon$, $\varepsilon \sim N(0, \sigma^2)$. For a test point $\tilde{x}$, $\tilde{y} = f(\tilde{x}) + \tilde{\varepsilon}$, $\tilde{\varepsilon} \sim N(0, \sigma^2)$

the expected test error can be written as

$$\mathbb{E}_{\mathcal{D}, \tilde{\varepsilon}}\left[(\tilde{y} - h_W(\tilde{x}; \mathcal{D}))^2\right]$$

shorthand for this, $\mathbb{E}\left[(\tilde{y} - h(\tilde{x}))^2\right]$

$$\mathbb{E}\left[(\tilde{y} - h(\tilde{x}))^2\right] = \mathbb{E}\left[\tilde{y}^2 - 2\tilde{y} h(\tilde{x}) + h(\tilde{x})^2\right] \underset{\text{of expectation}}{=} \mathbb{E}\left[h(\tilde{x})^2\right] + \mathbb{E}\left[\tilde{y}^2\right] - 2\,\mathbb{E}\left[\tilde{y}\right]\mathbb{E}\left[h(\tilde{x})\right]$$

$$= \text{Var}(h(\tilde{x})) + \mathbb{E}[h(\tilde{x})]^2 + \ldots$$

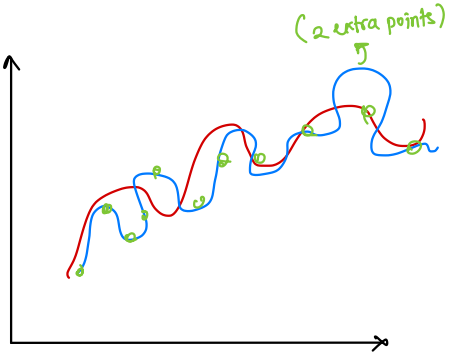
Note: $\tilde{y}$ is true value, h is predictor so are independent R.V.s, h varies as it's a f^n of dataset

$$= \mathbb{E}\left[(h(\tilde{x}) - \bar{h}(\tilde{x}))^2\right] + \mathbb{E}[h(\tilde{x})]^2 + \mathbb{E}\left[(\tilde{y} - f(\tilde{x}))^2\right] + f(\tilde{x})^2 - 2\,\mathbb{E}[h(\tilde{x})]f(\tilde{x})$$

$$= \left(\bar{h}(\tilde{x}) - f(\tilde{x})\right)^2 + \mathbb{E}\left[(h(\tilde{x}) - \bar{h}(\tilde{x})^2\right] + \sigma^2$$

↳ putting $\tilde{y} = f(\tilde{x}) + \tilde{\epsilon}$

$$= \boxed{\text{bias}^2 + \text{Variance} + \sigma^2}$$



In high complexity models, variance is high, bias is low.

In low complexity (say lines $\perp^{el}$ to x-axis), low variance, high bias.

Overfitting : High variance
Underfitting : High bias

For ridge regression, if λ increases, variance decreases and bias increases (modelling capabilities reduced).

$$w_{ridge} = \arg\min_{w} \|Y - Xw\|^2 + \lambda\|w\|^2$$

Lecture — Logistic Regression

$f: X \to Y = \{0,1\}$ (binary classification)

i.e. find the best deterministic estimate to a probability distribution over x $x\{0,1\}$. Loss f^n ?

1. Discrete loss (matrix L), $L_{ij} = \begin{cases} 1 & i \neq j \\ 0 & i = j \end{cases}$ ← common
2. $L_{ij} = |i - j|$ (absolute loss)
3. $L_{ij} = (i-j)^2$ (squared loss)

$$EPE(f) = \mathbb{E}_X\left[\mathbb{E}_{Y|X}\left[L(Y, f(X))\right]\right]$$

minimizing pointwise, we have the regression

$$f(x) = \arg\min_{y \in Y} \mathbb{E}_{Y|X=x}[L(Y, y)] = \arg\min_{y \in Y} \mathbb{E}_{Y|X=x}[\mathbb{1}[Y \neq y]]$$

$$= \arg\max_{y \in Y} P(Y = y \mid X = x)$$

i.e. we classify based on most likely label (Bayes' classifier)

we have $y = \{0, 1\}$, so, $P(Y=1 \mid X=x) = 1 - P(Y=0 \mid X=x)$.

So, we have $f(x) = \mathbb{1}_{[P(Y=1 \mid X=x) > 0.5]}$

Logistic Regression

we attempt to approximate with a linear f^n.

$$f(x) = \begin{cases} 1, & \text{if } \beta^T x > 0.5 \\ 0, & \text{if } \beta^T x \leq 0.5 \end{cases}$$

But $\beta^T x$ can be negative. So, we use sigmoid f^n

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$f(x) = \begin{cases} 1, & \text{if } \sigma(\beta^T x) > 0.5 \\ 0, & \text{if } \sigma(\beta^T x) \leq 0.5 \end{cases}$$

ML estimate for β

we have loss for dataset $D = \{(x_i, y_i)\}_{i=1}^n$

$$L = \log \pi_i P(y_i \mid x_i, \beta)$$

$$\log(y_i \mid x_i, \beta) = y_i \log P(y_i=1 \mid x_i, \beta) + (1-y_i) \log(1 - P(y_i=1 \mid x_i, \beta))$$

$$= y_i \log f(x) + (1-y_i) \log(1 - f(x))$$

So, $\beta_{ML} = \arg\min -\sum_i y_i \log f(x_i) + (1-y_i) \log(1-f(x_i))$

The loss, $L = -\sum_i y_i \log f(x_i) + (1-y_i) \log(1-f(x_i))$ {binary cross-entropy loss}

$$\mathcal{L} = -\sum_i y_i \log \sigma(\beta^T x_i) + (1-y_i) \log(1 - \sigma(\beta^T x_i))$$

$$\nabla_\beta \mathcal{L} = \sum_i (\sigma(\beta^T x_i) - y_i) x_i$$

No closed form solution for β.

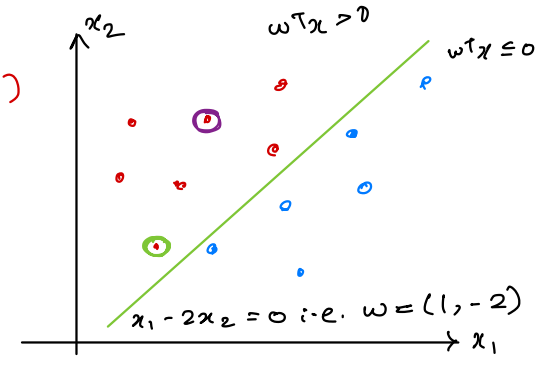
Lecture

31/8/23

Logistic Regression

$$w^* = \arg\max_w \sum_i \log P(y_i|x_i; w)$$

$$= \arg\min_w \text{Cross-entropy-loss}(w, D)$$



linearly separable data

Regularized Logistic Regression

$$w_R^* = \arg\min_w L_{CE}(w, D) + \lambda \|w\|_2^2$$

In the unregularised case,

purple should get higher probability of 1 than green point (near decision boundary)

$$P(y|x, w) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

If $w = (10^5, -2 \times 10^5)$ is an equivalent solution

Classifier will prefer high values of w to maximise $P(y|x, w)$.

But this will cause v. less difference b/w purple & green points.

Hence, regularisation is important to curb w, i.e. make variance smaller, bias ↑.

LR classifier: Fairly intuitive to interpret linear decision boundaries.

Not easy to do with non-linear classification boundaries (LR + feature transformation)

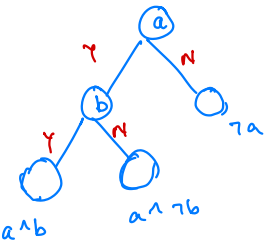
Desired:

1. The ability to learn complex decision boundaries
2. Interpretability of the model, i.e., how useful are each of the feature/attributes

Decision Tree (DT) classifiers

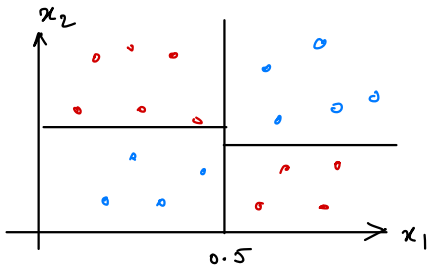
DT is an interpretable model whose final prediction can be written as a "disjunction of injunctions" based on attribute values over training instances.

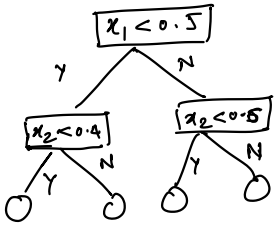
- Each path in the tree is a conjunction of attribute based constraints
- Tree: disjunction of the conjunctions



Tree $\equiv (a \wedge b) \vee (a \wedge \neg b) \vee (\neg a)$

Decision boundaries of decision trees





DT divides feature space into hyper-rectangles

≡ the resulting boundaries are axis ∥el hyperplanes

Typically, we work with single variable decisions. If we work with linear queries, we get general hyperplanes.

Finding the smallest optimal DT, i.e., optimal w.r.t. some metric involving all attributes ⇒ NP-hard.

DT estimation is largely greedy ⇒ recursively build the tree.

SIMPLE DT TEMPLATE

1. Start from an empty instance with all instances
2. Pick the best attribute to split on
3. Repeat step 2 recursively on each new node until a stopping criterion is met.

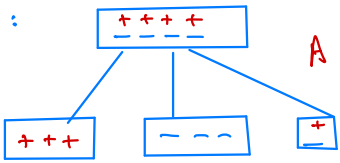
Two important questions

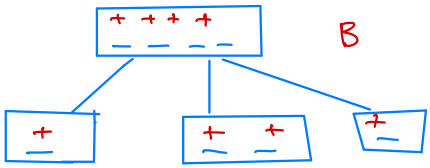
Q1 : What is the notion of a "best" attribute and how to find it ?

Q2 : What is a good stopping criterion ?

Tree shouldn't be too deep to avoid overfitting

Q1 :



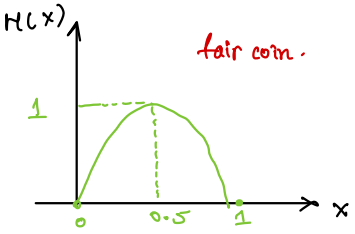


Intuition : A good split for an attribute results in subsets that are (nearly) all +ve or all -ve.

Information Gain

$$H(X) = -\sum_{x \in X} P(X=x) \log P(X=x)$$

default base 2



High entropy ⇒ nearly uniform distribution
⇒ high uncertainty

Low entropy ⇒ distribution has well-defined modes
⇒ low uncertainty

Entropy of a dataset S

$$H(S) = -\sum_i P_{i;S} \log P_{i;S}$$, where $P_{i;S}$ is the relative count of instances in S with label i

$$\text{Gain}(S, a) = H(S) - \sum_{v \in \text{values}(a)} \frac{|S_v|}{|S|} H(S_v)$$

(set of attributes)

where S_v is the subset S whose instances all have attribute a taking the value v.

Lecture

Information Gain of a feature X

$$IG(X) = H(Y) - H(Y|X)$$

$$H(Y|X) = \sum_x p(x) H(Y|X=x)$$

$$x^* = \arg\max_x IG(X) = \arg\min_x H(Y|X=x)$$

Few more measures of separation:

- Gini index = $1 - \sum p_i^2$, p_i = fraction of label i in data [total = weighted sum]
- Just entropy of the label.

When do we stop?

1. Stop when the data is pure — all elements in dataset are of the same label
2. Stop if number of instances at a node is less than some fixed k
3. Stop if maximum gain at a node ≤ threshold. (Used in conjunction with stop if the total number of leaves ≥ threshold)

Alternative to stopping criterion : pruning

Reduced error pruning = build tree till end and prune nodes that do not improve error on validation set (prune = remove subtree rooted at node)

{top down search}

Bootstrap aggregation or BAGging

Consider a dataset D of size n. Create M datasets of same size n from D by sampling with replacement. Bootstrapping - reduces variance. Now, train M models on these datasets. The final prediction is the majority vote over the M models. This is called bootstrap aggregation.

Random Forests

Special case of bagging - they are bagged decision trees. For each DT we also randomly select a subset of features to split the DTs on. This is done to reduce correlation between models - if there was a dominant feature, then all models would separate on this, and there would not be much point in many models. Irrelevant features in decision trees? The majority vote will take care of that.

04/09/23

Lecture

Perceptron Classifier

Linear models of classification. Goal is to estimate hyperplanes that best separate the training data.

Eqn of hyperplane $\Rightarrow w^T x = 0$

Perceptron classifier at test time predicts a test instance x to have the label $h(x) = \text{sign}(w^T x)$

where $\text{sign}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ -1, & \text{if } z < 0 \end{cases}$

Perceptron is:

1. A basic unit of the neural model of learning [Perceptron is very loosely based on a neuron]
2. Online: It makes updates by seeing training instances one at a time
3. Error-driven or mistake driven: weight updates are triggered only when the perceptron makes a mistake, does nothing otherwise

Perceptron algorithm

Inputs: training instances

Initialise weight vector as w^0, iteration counter $t=0$

for a fixed # of iterations, i.e., $t = 0 \ldots T-1$

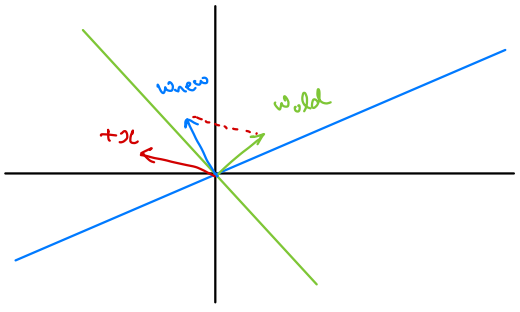
Pick a random training example (x', y') [or from a shuffled train set sequentially]

Predict $\hat{y} = \text{sign}(w_t^T x')$

if $\hat{y} \neq y'$

$w_{t+1} \leftarrow w_t + y' x'$

return w_T



Is the perceptron algorithm guaranteed to improve after a weight update on an erroneous training sample

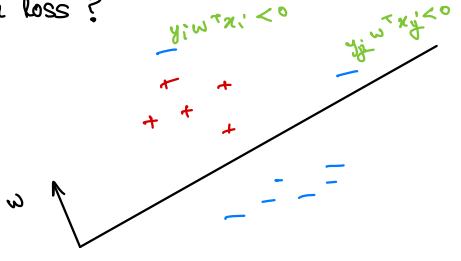
- Yes, but it is not guaranteed to become correct immediately after an update

$$y\, w_{new}^T x = y\,(w_{old} + y\,x)^T x$$
$$= y\, w_{old}^T x + y^2 \|x\|_2^2$$
$$> y\, w_{old}^T x$$

we know $y\, w_{old}^T x < 0$ and $y\, w_{new}^T x > y\, w_{old}^T x$ thus making it closer to a positive value.

Since sign is encoded in $y = \pm 1$

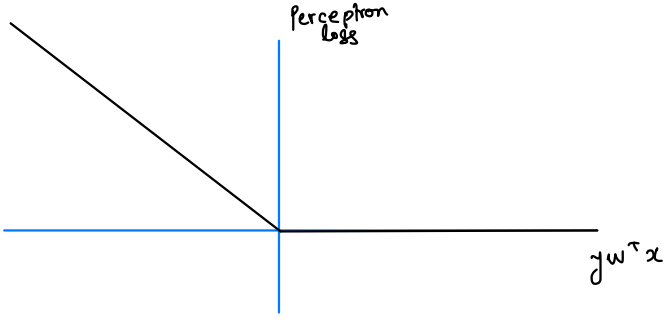
What is the perceptron algorithm minimizing as a loss?



Minimize the number of misclassified examples

$$\Rightarrow \min \sum_{i \in \{\text{indices of misclassified e.g.}\}} -y_i w^T x_i$$

Perceptron loss of $(x, y) = \max\{0, -y w^T x\}$



$$L_{perceptron}(w, D) = \sum_i \max\{0, -y_i w^T x_i\}$$

Run SGD on perceptron loss to find w. For this compute subgradients.

A subgradient of a convex fn f at w_0 is all vectors g s.t. for any other point w, we have $f(w) - f(w_0) \geq g^T(w - w_0)$

Lecture

11/09/23

Consider a linearly separable dataset D, i.e. one such that there exists a (unit) vector u such that $Y = \text{sgn}(u^T x)$, for every $x, y \in D$.

Without loss of generality, $\forall x_i \in D, \|x_i\| \leq 1$.

Define margin of separation D w.r.t. u be defined as

$$\gamma = \min_{x \in D} |u^T x|$$

that is, γ is the minimum distance of pt $x \in D$ from hyperplane $u^T x = 0$.

Thm: If $\exists u$ s.t. $(x,y) \in D$, we have $yu^T x \geq \gamma$. Then, perceptron algorithm will make no more than $\frac{1}{\gamma^2}$ mistakes on the training dataset when initialized with $w = 0$.

Proof: w appears to converge to 0 loss vector

$$w_{i+1} = w_i + yx$$

$$w_{i+1}{}^T u = w_i{}^T u + yx^T u \geq w_i{}^T u + \gamma$$

$$\|w_{i+1}\|^2 = \|w_i\|^2 + 2yw^T x + \|x\|^2 \leq \|w_i\|^2 + 0 + 1 = \|w_i\|^2 + 1$$

$$\Rightarrow \|w_k\| \leq \sqrt{k} \quad \forall k.$$

If perceptron algorithm makes k mistakes,

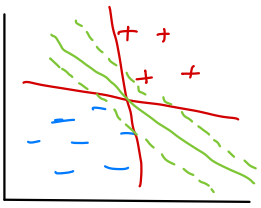
$$\sqrt{k} \geq \|w_k\| \geq w_k{}^T u \geq k\gamma \Rightarrow k \leq \frac{1}{\gamma^2}$$

M2: $y w_{i+1}{}^T x = y w_i{}^T x + \|x\|^2 \leq y w_i{}^T x + 1$

$$\max(0, -y w_{i+1}^T x) \leq \max(0, -y w_i{}^T x) - 1$$

Lecture (07/09/23)

Support Vector Machines



SVMs are max-margin classifier that aim to maximize the margin while correctly classifying all the training points.

Here, margin is the distance of the closest point across both classifiers from the hyperplane.

$$\text{Margin } \gamma(w,b) = \min_i \frac{|w^T x_i + b|}{\|w\|} \rightarrow (A)$$

SVM wants to maximize the margin & correctly classify all training pts

$$\max_{w,b} \; 2\gamma(w,b) \quad \text{s.t.} \quad \forall i \; y_i(w^T x_i + b) \geq 0 \rightarrow (B)$$

(2 — for a nice gradient)

Substituting (A) in (B),

$$\max_{w,b} \frac{2}{\|w\|} \min_i |w^T x_i + b| \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 0 \rightarrow (C)$$

Fix the scale of w,b s.t. $\min_i |w^T x_i + b| = 1 \rightarrow (D)$

Note: This is allowed as $d(x,L) = \frac{\|L(x)\|}{\text{(scale)}}$ ← (scale = sum of w_i^2)

From (C) and (D),

$$\max_{w,b} \frac{2}{\|w\|} \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 0, \quad \min_i |w^T x_i + b| = 1$$

$$\equiv \min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 0, \quad \min_i |w^T x_i + b| = 1 \rightarrow (E)$$

Hard margin sum optimization problem

From (E),

$$\min_{w,b} \frac{1}{2}\|w\|^2$$
$$\text{s.t. } y_i(w_i^T + b) \geq 1$$

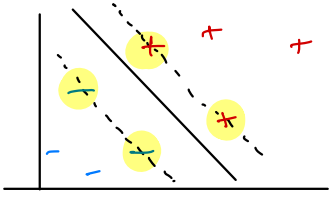
Note: Equivalent to E ! (why ?)

$y_i \in \{-1, 1\}$. Hence, $y_i(w^T b + 1) \geq 1$

$\Rightarrow |w^T b + 1| \geq 1.$

$\Rightarrow \min |w^T b + 1| \geq 1.$

Why hard margin ? Because we want every training example to satisfy margin constraint



Highlighted points = tight constraints. If not, we can rescale w,b s.t. constraints become tight.

The tight points are "support vectors".

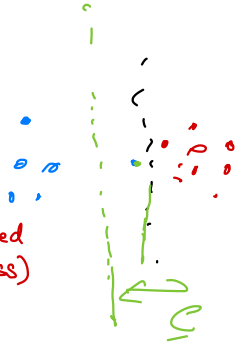
Not Linearly Separable data

⇒ This means that ∃ some pts which violate margin constraints

$\exists_i \; y_i(w^T x_i + b) < 1$ {given non-lin sep. data for any w}

$\Rightarrow \exists_i \; (1 - y_i(w^T x + b)) > 0$

Constraint violation = $\begin{cases} 0 & \text{if } 1 - y_i(w^T x + b) \leq 0 \\ 1 - y_i(w^T x + b) & \text{otherwise.} \end{cases}$



Minimize the total violation of margin across D_i:

$\sum_i \max\{0, 1 - y_i(w^T x_i + b)\}$ (Hinge loss)

Revised unconstrained SVM optimization problem

$\min_{w,b} \frac{1}{2}\|w\|^2 + C \sum_i \max\{0, 1 - y_i(w^T x_i + b)\}$ (Regularised Hinge Loss)

Soft-margin Sum optimization problem

Equivalent constrained opt. to (A)

$\min_{w,b} \frac{1}{2}\|w\|^2 + C \sum_i \xi_i$ → slack variables.

s.t. $\forall_i \; y_i(w^T x_i + b) \geq 1 - \xi_i$

$\forall_i \; \xi_i \geq 0$

What is the role?

C is a penalty or regularization factor. Balances the trade-off between large margins and large-training error.

SVM Optimization Problem

$\min_{w,b} \frac{1}{2}\|w\|^2$

s.t. $y_i(w^T x_i + b) \geq 1$

Define $g_i(w) = 1 - y_i(w^T x_i + b) \leq 0$

For each constraint, define a lagrange multiplier $\forall_i \; \alpha_i \geq 0$, write Lagrangians.

$L(w, \alpha) = \frac{1}{2}\|w\|^2 + \sum_i \alpha_i (1 - y_i(w^T x_i + b))$

Lecture

11/09/23

Generic constrained optimization problem is :

$$\min_{w} f(w) \quad \text{(Primal)}$$

$$\text{s.t. } g_1(w) \le 0, \dots, g_n(w) \le 0$$

Lagrangian $L(w, \alpha) = f(w) + \sum \alpha_i g_i(w)$

Lagrange mult

Equivalent formulation of the primal is

$$\min_{w} \max_{\alpha \ge 0} L(w, \alpha)$$ [removing constraints on w]

Why the equivalence ?

$$\max_{\alpha \ge 0} f(w) + \sum \alpha_i g_i(w) = \begin{cases} f(w) & \text{if } w \text{ satisfies all constraints} \\ \infty & \text{otherwise} \end{cases}$$

Consider the dual problem

$$d^* = \max_{\alpha \ge 0} \min_{w} L(w, \alpha)$$

In general $d^* \le p^*$.

For the SVM problem, $\boxed{d^* = p^*}$

Dual for SVM

$$\max_{\alpha \ge 0} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$\text{s.t. } \forall_i \ \alpha_i \ge 0, \ \sum \alpha_i y_i = 0$$

(Hard-margin SVM)

$$L(w, \alpha) = \frac{1}{2}\|w\|^2 + \sum_i \alpha_i (1 - y_i (w^T x_i + b))$$

$$\nabla_w L(w, \alpha) = w + \sum \alpha_i (-y_i x_i) = 0$$

$$\Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

← put in $L(w, \alpha)$

Similarly $\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$

In soft-margin SVM,

$0 \le \alpha_i \le C$.

Everything else is same as hard margin

observations based on the dual form,

① $w = \sum_{i=1}^{n} \alpha_i y_i x_i$ ← $\alpha_i > 0 \Rightarrow$ tight constraints, i.e. sv, hence w depends only on support vectors.

② If the solution to primal is w^* and to dual is some α^* then KKT conditions on w^* and α^* hold for both SVM problems.

One of the dual complementality constraint, is of interest

$$\boxed{\alpha^*_i \, g_i(w^*) = 0 \quad \forall i}$$

$\alpha_i > 0$ corresponds to those training examples for which $g_i(w)$ is the equality, i.e; $g_i(w) = 0$.

$$\Rightarrow \quad y_i (w^T x_i + b) = 1$$

These are the "support vectors"

Dual Hard-margin SVM

$$\max_{\alpha} \; \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j \;, \quad \alpha \geq 0, \; \sum \alpha_i y_i = 0$$

③ The dual form operates only on inner products of training examples pairs.

Test Time

$$\text{Prediction} = \text{sgn}(w^{*T} x + b^*)$$

$$= \text{sgn}\left(\left(\sum_i \alpha_i^* y_i x_i\right)^T x + b^*\right)$$

$$= \text{sgn}\left(\sum_i \alpha_i^* y_i \underbrace{x_i^T x}_{\text{again, inner product}} + b^*\right)$$

If we want a non-linear decision boundary using SVMs,

$$\text{Prediction} = \text{sign}\left(\sum_i \alpha_i^* y_i \underbrace{\phi(x_i)^T \phi(x_i)}_{\text{feature transform } \phi : \mathbb{R}^d \to \mathbb{R}^m} + b^*\right)$$

Two problems :

1. How do I find $\phi(x)$?
2. Computationally expensive to calculate $\phi(x)$ and compute dot product

Kernels

Consider a kernel function $K(x,y) = \phi(x)^T \phi(y)$

If we can compute $K(x,y)$ without computing $\phi(x)$, then this is useful (kernel trick)

Consider the following example, for instances $x = (x_1, x_2), y = (y_1, y_2)$

$$\phi(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{bmatrix}.$$

Define a kernel function $K(x,y) = (x^T y)^2$

Verify that $K(x,y) = \phi(x)^T \phi(y)$

Can we use any function for K? No

(1) A kernel $K(x,y)$ is valid kernel function of it corresponds to an underlying $\phi(x)$ [projecting x to a

That is, $K(x,y) = \phi(x)^T \phi(y)$

(2) Let $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a valid kernel, the necessary and sufficient condition is that for any finite set $\{x_1, \ldots, x_n\}$ the kernel matrix $[K(x_i, x_j)]$ is symmetric and positive semi-definite [Mercer's theorem]

(3) Use properties of kernels over existing valid kernels

$$K(x,y) = K_1(x,y) + K_2(x,y)$$

$$K(x,y) = K_1(x,y)\, K_2(x,y)$$

$$K(x,y) = \alpha K_1(x,y), \alpha > 0$$

Examples of valid kernels

1. Polynomial kernel $K(x,y) = (x^T y)^d$ [or $(c + x^T y)^d$: degree $\leq d$]
2. Radial basis f^n / Gaussian kernel : $K(x,y) = e^{-\frac{\|x-y\|^2}{\sigma^2}}$

$K(x,y)$: measure of similarity b/w pairs of points

Q: Is $K(x,y) = \|x+y\|^2$ a valid kernel?

$K(x,y) = (x+y)^T (x+y)$

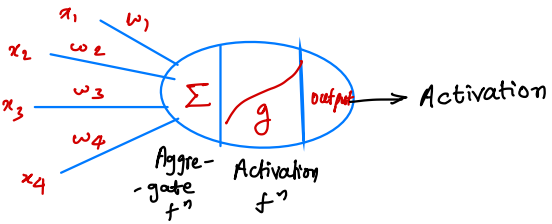
$K(0,0) = 0 \Rightarrow \phi(0) = 0$.

$\Rightarrow K(x,0) = 0$, but $K(x,y) = x^T x$. (contradiction)

12/09/23

Lecture

McCulloch-Pitts Neuron Model

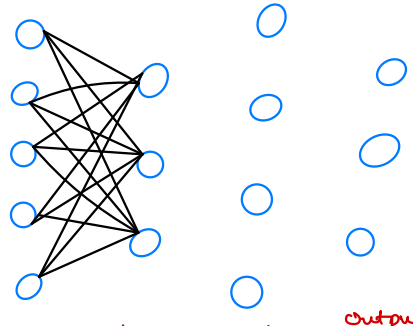


Limitation : classification of XOR

Neuron activation $= g\left(\sum_{i=1}^{n} w_i x_i\right)$

$\Sigma = \Sigma$, g = sign $\longrightarrow$ Perceptron.

Let's stack neuron models together to form a multi-layered neural network



Input layer — $x \in \mathbb{R}^d$

1st hidden layer — $y \in \mathbb{R}^{n \times 1}$

Hidden layer — $z \in \mathbb{R}^{m \times 1}$

Output layer — $o \in \mathbb{R}^{c \times 1}$

$W \in \mathbb{R}^{n \times d}$ $V \in \mathbb{R}^{m \times n}$ $U \in \mathbb{R}^{c \times m}$

W is an $n \times d$ matrix where w_{ij} refers to weight of connection from i^{th} row in input layer to j^{th} row in hidden layer

$y = g(Wx + b)$, b = bias setting

↑ applied elementwise

$z = g(Vy + b') = g(V\, g(Wx+b) + b')$

$o = g(Uz + b'')$

Fully-connected (or) feed-forward network (not necessarily fully connected, but this is default)

Output $o \in \mathbb{R}^{c \times 1}$: What are useful loss functions?

① Regression : Squared error loss $= \sum_i \frac{1}{2}(o_i - y_i)^2$

② Classification : A very handy transformation is the SOFTMAX operation.

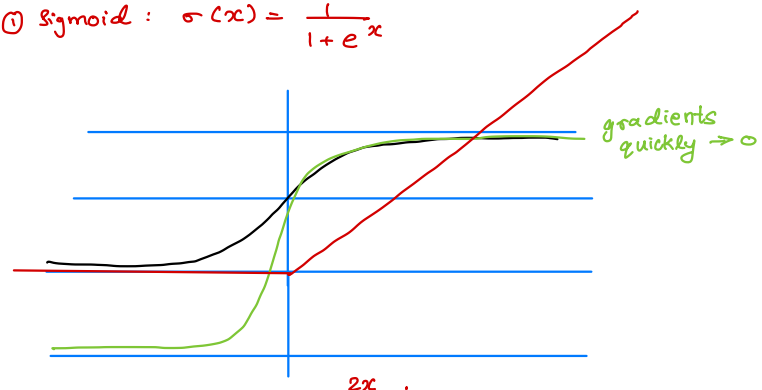
$$\text{SOFTMAX}(o_j) = \frac{e^{+o_j}}{\sum_{k=1}^{c} e^{+o_k}}, \quad j \in \{1, \ldots, c\}$$

{generalisation of sigmoid}

$P_{NN}(y|x) = \text{SOFTMAX}(o)$ {then use cross-entropy loss on this}

Common Activation Functions (g)

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



② tanh : $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$ outputs b/w −1 and +1

③ RELU : $relu(x) = \max(0, x)$ Does remarkably well. Simple gradients, converges faster

---

Loss function $f(w, D)$

(Credits : KNA)

Typical loss function

For regression problems (the output vector is scalar), with true output y, output of MLP (multi-layer perceptron) $\hat{y}$,

$$L(y, \hat{y}) = \frac{1}{2}(y-\hat{y})^2$$ (squared error loss)

For classification problems, cross-entropy loss is typically used.
Suppose we have k-classes, output vector of MLP is k-dimensional.
Let y be the true label of the input x, let $\hat{y}$ be output of MLP on input x

$$L(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$ (cross-entropy loss)

A trick to convert output as probabilities

$$f(y=i|x) = \frac{e^{o_i}}{\sum_{j=1}^{k} e^{o_j}}$$

where $(o_1, \ldots, o_k)$ is the output of the network on input x.

Backpropogation

To obtain weights as function of training data, we use regular SGD.

Initialize weights and biases randomly

while not converged do

Sample a minibatch of data $\mathcal{D}$

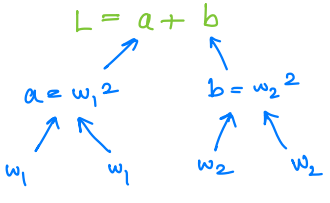
Compute the gradient of the loss function $\nabla_w L$ on $\mathcal{D}$ with respect to weights

Update the weights, biases using gradient $w \leftarrow w - \alpha \nabla_w L$

end while

Computation of gradients ? Use backpropogation.

We can write dependence of loss on weights as directed acyclic graph of loss on the weights e.g. $L(w_1, w_2) = w_1^2 + w_2^2$



Here, $g(a,b) = a+b$, $a = w_1^2$

we get $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial w_1}$

$= 1 \cdot 2w_1$

similarly $\frac{\partial L}{\partial w_2} = 2w_2$

Thm (Multivariable Chain Rule)

Suppose $f = f(y_1, \ldots, y_n)$ is a function. For some i, suppose $y = g(x_i)$

Then, $\frac{\partial f}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial y_j}\frac{\partial y_j}{\partial x_i}$

So, the idea is we can compute $\frac{\partial L}{\partial w}$ if we know $\frac{\partial u}{\partial w}$ for every child u., and form a recursive procedure.

Algorithm

Algorithm has a forward pass, backward pass. Forward pass computes values of nodes in the DAG. The backward pass computes derivates of the loss with respect to weights (i.e. gradients)

Forward pass Compute the values of nodes in DAG in topological order

Algorithm 2 forward pass

1: The values of the leaves are the input values and the values of the weights.
2: **for** each node u in the DAG in topological order **do**
3: compute $u.val = f(u.child_1.val, u.child_2.val, \ldots, u.child_k.val)$
4: **end for**

Backward pass

Here, we use the computed values to compute gradient of every node with respect to the leaves (s.t. 0 i.e. we don't care abt $\frac{\partial L}{\partial x}$)

While building the DAG, we store dependencies of the parent $di = \frac{\partial u}{\partial (u.child_i)}$ on its children's functions. We use some of input vals (e.g. $w^T x$), but we do not include them in children because they aren't vars we're differentiating with respect to.

Algorithm 3 backward pass

1: The function dw for the leaves (weights) is set to 1.
2: **for** each node u in the DAG in topological order **do**
3: compute $u.grad = \sum_{i=1}^{k} u.di \times child_i.grad$ (where $\star.grad$ is a vector with dimension equal to the number of weights)
4: **end for**

Regularization

1. L2 regularization : Add term $\lambda \|w\|_2^2$ to loss

2. Dropout :

During training, randomly set the activations of some neurons to 0. This prevents network from relying too much on any one neuron, and thus prevents overfitting.

- Do not drop out input, output neurons
- At test time, multiply activations of each neuron by probability that it was dropped out during training, i.e. expected value of activation is used.

Remark : An extreme version of dropout is to randomly drop out entire layers of neurons (called drop connect). This makes use of Residual connections, which connect the input of a layer to the output of a layer that is not the immediate previous layer.

3. Early stopping :

Stop training when the validation loss starts increasing. This is a form of regularisation because it prevents network from overfitting to training data (we can stop training before it can do so).

Optimizers

1. Momentum

- Large weight updates in complex loss landscapes makes training unstable
- At flat points like saddle points, SGD gets stuck

SGD with momentum addresses these issues.

$\nabla_w L(w_t)$ means $\nabla_w L$ at $w = w_t$, it's not multiplication :)

Let $0 \le \beta < 1$ and define velocity v_t as

$$v_t = \eta (\nabla_w L)(w_t) + \beta v_{t-1}$$

So,

$$\begin{aligned} v_t &= \eta (\nabla_w L)(w_t) + \beta \left(\eta \nabla_w L(w_{t-1}) \right) + \beta^2 v_{t-2} \\ &= \dots \\ &= \eta \sum_{i=0}^{t} \beta^i \nabla_w L(w_{t-i}) \end{aligned}$$

and weight update is,

$$w_{t+1} = w_t - v_t$$

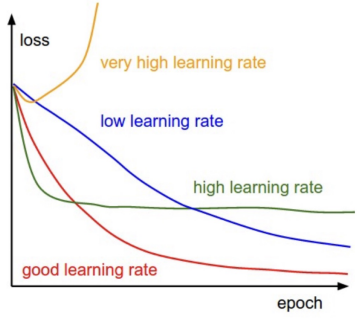
Note: This escapes flat points, but will escape minima as well, then come back as velocity reduces, velocity will likely be small upon coming back so won't overshoot again

2. ADAgrad (adaptive gradient)

- Have different learning rate for each parameter as some parameters are more sensitive than others

$\odot \equiv$ element-wise multiplication

$$s_t = s_{t-1} + g_t \odot g_t \quad , \quad g_t = \nabla_w L(w_t)$$



and, weight update is,

$$w_{t+1} = w_t - \eta \frac{1}{\sqrt{s_t} + \varepsilon} \odot g_t$$

So, we get

$$\eta_i = \frac{\eta}{\sqrt{\sum_{t=1}^{\tau} (\nabla_w L(w_t))^2} + \varepsilon}$$

, weight update $(w_{t+1})_i = (w_t)_i - \eta_i (\nabla_w L(w_t))_i$

Issue: learning rate has sum of squares in denr, so algorithm eventually stops learning. This issue is solved by the next optimizer

3. RMS Prop

$$s_t = \gamma s_{t-1} + (1-\gamma)\, g_t \odot g_t$$

Note : This allows us to forget old gradients. This was proposed in a Courtera course by Goeff Hinton !

4. ADAM

Here, we combine the above ideas.

$$v_t = \beta v_{t-1} + (1-\beta)\, g_t$$

$$s_t = \gamma s_{t-1} + (1-\gamma)\, s_t \odot s_t \quad \text{, where } g_t = \nabla_w L(w_t)$$

update step : $w_{t+1} = w_t - \dfrac{\eta}{\sqrt{\hat{s}_t + \varepsilon}} \odot \hat{v}_t$

where $\hat{s}_t = \dfrac{s_t}{1-\gamma^t}$, $\hat{v}_t = \dfrac{v_t}{1-\beta^t}$ } to correct the fact that, v_t, s_t initialised to 0, so they are biased toward zero.

We have,

$$v_t = (1-\beta) \sum_{i=0}^{t-1} \beta^i g_{t-i} \quad \text{, with sum of coeff being } 1-\beta^t,$$

so it helps to make velocity directly a moving avg of gradients for small t.

Lecture (CNNs) 26/09/23

Optimizers

A. SGD with momentum : smoothed fast gradients over time

B. ADAGrad : Adaptive learning rates per feature

C. RMSProp

D. Adam : Combination of A and B

ADAM $s_t \leftarrow \gamma s_t + (1-\gamma) g_t \odot g_t$

$v_t \leftarrow \beta v_{t-1} + (1-\beta) g_t$

$\hat{s}_t = \frac{s_t}{1-\gamma^t}$, $\hat{v}_t = \frac{v_t}{1-\beta^t}$ → Bias correction

Wt update for Adam $w_t \leftarrow w_{t-1} - \frac{1}{\sqrt{s_t} + \varepsilon} \odot \hat{v}_t$

$s_t = (1-\gamma) g_t \odot g_t + \gamma s_{t-1}$

$= (1-\gamma) g_t \odot g_t + (1-\gamma)\gamma\, g_{t-1} \odot g_{t-1} + \cdots + (1-\gamma)\gamma^t s_0$

Sum of coeff of s_t : $(1-\gamma)(1+\gamma+\cdots) \approx 1$ (if t is large)

for small t, sum $= 1-\gamma^t$

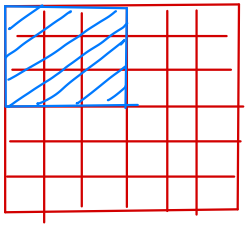
So, $\frac{s_t}{1-\gamma^t}$ mitigates the defect.

Consider the problem of recognising an object in an image

DESIDERATA	FFN	CNNs
(A) Locality	X (fully connected)	✓
(B) Translation invariance	X (different connections)	✓
(C) Parameter efficient	X (expensive)	✓

FFN = feed forward NN

Main operation underlying CNNs is convolution.



Window multiplication = "cross-correlation" but not convolution.

For this purpose, we loosely use the term convolution.

(A) Image ⊛ $\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$ = Blurred image

Mean Kernel

Monochromatic → no change

Variety → "average"

(B)

$$\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$ = Detects vertical edges

Monochromatic = zero

Sobel filter ≡ edge-detection filter

CNNs let you learn the kernel weights from data

Ref: Conv slides

05/10/23

Batch Normalisation

Recall that preprocessing/standardizing the inputs to have zero mean & unit variance was a typical first step. Why?

1. Constrains the function complexity of the hypothesis in $\mathcal{H}$
2. Standardization works well with GD-style optimizers by putting parameters apriori in a smaller space.

In deep NN can we add normalization within network

Batch Normalization ('15)

Variables can have val with largely varying magnitude.

BN authors conjectured that such a drift in distribution of internal variables (dubbed "internal covariate shift") can cause convergence challenges in deep networks

neurips '18: just ensures smoothness, cleaner updates (no covariate shift)

(worse covariate shift, smoother $\Rightarrow$ better acc %.)

Adaptive solvers (e.g. Adagrad) aim to address this issue. Alternate is to pre-emptively prevent this from happening (e.g. Batch Norm)

Batch Norm has 2 steps

1. In each training iteration, from the activations, subtract mean and divide by std-deviation. Both mean, & computed using batch level statistics
2. Next, apply scaling coeff, shift by offset (learnable)

Let B denote the batch, $x \in B$ then

$BN(x)$ can be written as

$$BN(x) = \gamma \odot \frac{x - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

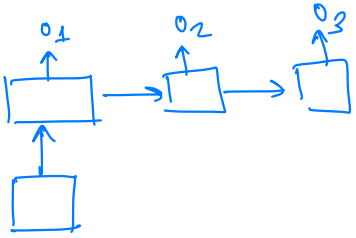
(γ, β: learnable; $\odot$: elementwise product)

$$\hat{\mu}_B = \frac{\sum_{x \in B} x}{|B|}, \quad \hat{\sigma}_B = \frac{1}{|B|} \sum_{x \in B} (x - \mu_B)^2 + \varepsilon$$

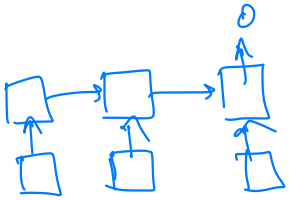
At test time: Use running avgs of mean, s.d. values during training

Note: BN is applied after affine, before non-lin. activation

(BN is linear, but causes "smoothness" in traversal of loss landscape)

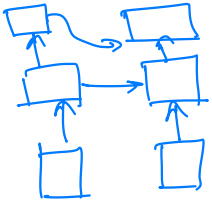


fixed input
variable length output
e.g. image descriptor



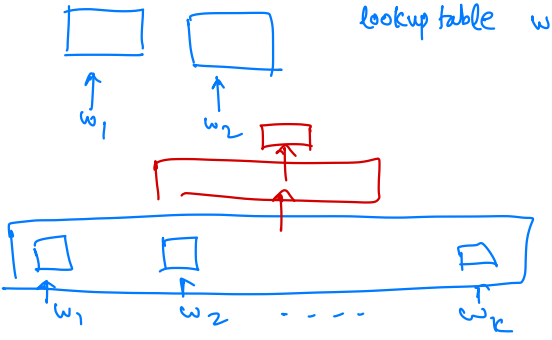
fixed o
variable length input
e.g. sentiment analysis

Both variable length (e.g. sequence labelling problem (e.g. translation). output length = f(input len))



Language Modelling (Causal language problem)

Given a world history $w_1, \ldots, w_{t-1}$ what is $w^* = \arg\max_w P(w | w_1 \ldots w_{t-1})$?

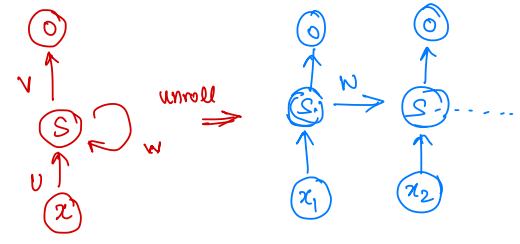


lookup table $w \to \mathbb{R}^d$ (word embedding)

NN

Deep NLP

Recurrent NNs



$s_t = \tanh(Ux_t + Ws_{t-1} + b)$

$o_t = Vs_t + b$

$\hat{y}_t = \text{softmax}(o_t)$

RNNs by design model temporal dependencies

Challenge: Vanishing and exploding gradients.

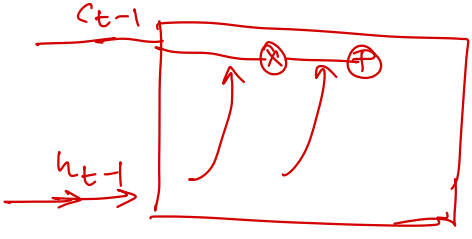
Recall that gradients in earlier layers = prod of grad in later layers (exploding gradients)

Band-aid: clip gradients $\leq$ threshold

If latter takes small vals: vanishing gradients

Long short term memory (LSTM) network

In LSTMs, the recurrent unit is enhanced with a cell state (or a memory state) and a number of learnable gates



forget gate:

$$f_t = \sigma(w_f [x_t; h_{t-1}] + b_f)$$

(element wise)

$\to 0$: forget

$\to 1$: retain

input gate: $i_t = \sigma(w_i [x_t, h_{t-1}] + b_i)$

modified cell state: $\tilde{c}_t = \tanh(w_c [x_t, h_{t-1}] + b_c)$

Final cell state: $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$

Output gate: $o_t = \sigma(w_o [x_t, h_{t-1}] + b_o)$

hidden state: $h_t = o_t \odot \tanh(c_t)$

GRU: gated recurrence unit.

05/10/23

Clustering

Consider a set of points $x_1, \ldots, x_n \in \mathbb{R}^d$. Assume that each datapoint x_i belongs to one of K clusters (we want points within each cluster to be "similar", i.e. close to one another in terms of euclidean distances)

How do we identify such clusters of datapoints?

E.g. Stock market indices, demographics, social networks

Motivation:

(A) Prediction based on labelling single point

(B) Outlier detection

(C) (Lossy) Compression

K-means clustering

Main intuition: Assume that there are K clusters. Every datapoint is close to the centre or the mean of its assigned cluster.

If we fix the cluster centres, its easy to make assignments for every datapoint.

If we fix assignments, easy to compute cluster centres.

(Chicken & egg problem)

fix one, optimize for the other and alternate.

Note: This is an "alternating minimization algorithm".

K-means objective

Find cluster centres $\mu_1, \ldots, \mu_K \in \mathbb{R}^d$ and assignments $c^1, \ldots, c^n$ [where each c^i is a one hot encoding of size K] such that sum of squared distances of all datapoints $x_1 \ldots x_n \in \mathbb{R}^d$ from their assigned cluster centres is minimized

formally: $$\sum_{i=1}^{n} \sum_{k=1}^{K} c_k^i \|\mu_k - x_i\|^2$$

where $c_k^i = \mathbb{1}[x_i \text{ assigned to cluster } K]$

"best assignment"

k-means Heuristic : Fix one, optimize the other

Step 1 : Fix μ, optimize the assignment.

For the i^{th} datapoint, $\sum_{k=1}^{K} c_k^i \|\mu_k - x_i\|^2$

$$c_k^i = \begin{cases} 1 & \text{, if } k = \underset{j}{\operatorname{argmin}} \|\mu_j - x_i\|^2 \\ 0 & \text{, otherwise} \end{cases}$$

Step 2 : Fix c, optimize cluster centres

For $k = 1, \ldots, K$

$$\frac{\partial}{\partial \mu_k} \sum_i \sum_k c_k^i \|\mu_k - x_i\|^2 = 0$$

$$\Rightarrow \mu_k = \frac{\sum_i c_k^i x_i}{\sum_i c_k^i}$$

{i.e. centroid of cluster points}

K-means Algorithm

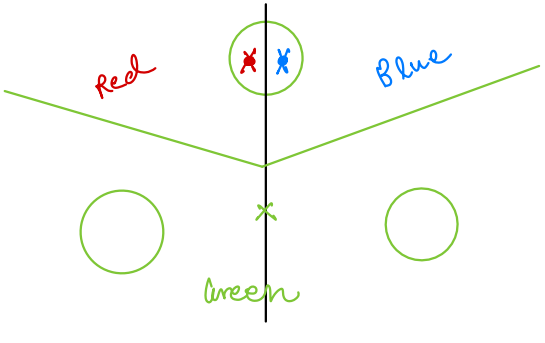
Initialize the cluster centres randomly $(\mu_1, \ldots, \mu_K)$

Repeat until convergence [i.e. assignments do not change]

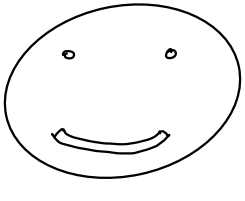
Assignment : Assign each datapoint to nearest cluster

Update : Recompute the cluster centres as the mean of all datapoints assigned to a cluster.

Note : This is not guaranteed to converge "correctly"



X = initialisation.



← no good k-means using alternation algorithms.

Note : K-means is NP-hard.

Q. Is K-means guaranteed to ~~converge~~? terminate.

K-means reduces the cost at every iteration (update as well as assignment, both local optima)

This, combined with the fact that number of cluster assignments is finite (since finite set of pts).

Hence, cycles cannot happen in configurations.

Convergence (termination) is guaranteed.

So far,

K-means is a "HARD" assignment meaning each datapoint is assigned to a single cluster

Soft assignment ? (probabilistic)

$$c_k^i = \frac{\exp(-\beta \| \mu_k - \mu_i \|^2)}{\sum_j \exp(-\beta \| \mu_j - \mu_i \|^2)}$$

Note : $\beta \to \infty$ then this is the hard K-means problem.

Lecture

10/10/23

Abstract alg.

θ = assignments (ref: abstract code)

ϕ = cluster centres

1. Both updates cost decreases, so cost always decreases

* 2. True

3. False. θ_0 can be revisited. θ can be repeated atmost twice

4. θ is finite, can be repeated atmost twice, so always terminates.

Recap: Hard K-means vs Soft K-means

Generative/Probabilistic Model of Clustering

Imagine that datapt $\mathcal{D} = \{x_1, \ldots, x_N\}$ are sampled from a generative model.

Idea: Once you have $P(X)$, we can generate joint distributions.

NNs : discriminative model.

Generative AI : (Andrew Ng : logistic regression & its generative counterpart) ← nice reference. (add" reading)

Note : Generative Models vs Discriminative Models

Model of $P(X)$ [or class conditional generative models $P(X|Y)$]

Models of $P(Y|X)$

Note: You can use $p(x|y)$, $p(y)$ for discriminative models (bayes rule)

Then, estimate the parameters of this generative model using MLE

Clustering using a Generative Model

Consider a datapoint x from the dataset $\mathcal{D}$

- Pick a cluster z from $\{1, \ldots, K\}$ s.t. $P(z=k) = \pi_k$
- Given a z, sample from a gaussian distribution $P(x|z) = \mathcal{N}(x; \mu_z, I)$

[Note: For simplicity of subsequent derivations, we'll assume $\Sigma = I$; more generally, $\Sigma = \Sigma_z$]

Generative Process:

$$P(z=k) = \pi_k, \quad \sum_{k=1}^{K} \pi_k = 1$$

$$P(x \mid z=k) = \mathcal{N}(x; \mu_k, I)$$

How do we estimate $\{\pi_k, \mu_k\}_{k=1}^{K}$?

Use MLE (max. likelihood estimation)

$$\max \log p(x) = \log \sum_k p(z=k, x)$$

(x: observed vars; z: hidden variable)

$$= \log \sum_k p(x|z=k)\, p(z=k)$$

$$= \log \sum_k \pi_k \mathcal{N}(x; \mu_k, I)$$

($p(x)$: single datapoint)

mixture component weights / mixing coefficients

$$p(x) = \sum_k \pi_k \mathcal{N}(x; \mu_k, I)$$

Gaussian mixture model (GMM)

Aside: Note on mixture models.

Going back to the objective to maximize for generative model

$$\sum_{i=1}^{N} \log p(x_i) = \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, I)\right)$$

This is cumbersome to optimize.

So, let's simplify to a hard cluster setting, i.e, for every x_i, assume you know corresponding z_i (z_i are also given) ← Fully observable setting

$$\max \sum_i \log p(x_i) = \sum_i \log \sum_k p(z=k, x_i)$$

To do: fill in steps

$$= \max \sum_i \sum_k \mathbb{1}(z_i=k)\left(\log \mathcal{N}(x_i, \mu_k, I) + \log \pi_k\right) - (A)$$

Do MLE now for μ_k, π_k for (A)

$$\mu_k = \frac{\sum_i \mathbb{1}[z_i=k]\, x_i}{\sum_i \mathbb{1}[z_i=k]}, \quad \pi_k = \frac{1}{N} \sum_i \mathbb{1}[z_i=k]$$

To do: fill in steps

Now, we build up to a solution where we don't know what z_i's are.

Consider the objective in (A)

we do not have information about $\mathbb{1}[z_i = k]$, but, we can compute the $\mathbb{E}[\mathbb{1}[z_i=k]] = P(z_i = k \mid x_i)$ ← # To do: $\mathbb{E}$ over what? fill in explanation

So, compute $P(z_i = k \mid x_i) = \dfrac{P(x_i \mid z_i = k)\, P(z_i = k)}{P(x_i)}$ [refer to this as $r_{k,i}$]

$$= \frac{\mathcal{N}(x_i; \mu_k, I)\, \pi_k}{\sum_j \pi_j\, \mathcal{N}(x_j; \mu_j, I)}$$

Looking at expression in (A), replace indicator with $r_{k,i}$ to get:

$$\sum_i \sum_k r_{k,i} \left(\log \mathcal{N}(x_i, \mu_k, I) + \log \pi_k\right) \quad - (B)$$

Get (B) from (A) by assuming $r_{k,i}$ are fixed, meaning they use current values of parameters μ_k, π_k

Note: chicken-egg situation once again, to get r we need $\mathbb{1}$. and vice-versa. $r \leftrightarrow \pi_k, \mu_k$

This motivates the expectation maximization algorithm [EM algorithm]

Lecture 12/10/23

Recall the fully observable setting, the MLE

$$\sum_i \log P(x_i, z_i) = \sum_i \sum_k \mathbb{1}[z_i = k] \left(\log \pi_k + \log \mathcal{N}(x_i; \mu_k, I)\right)$$

In the real setting only $x_1, \ldots x_N$ are observed, z_i's are hidden / latent

$$r_{k,i} \triangleq P(z_i = k \mid x_i) = \pi_k\, \mathcal{N}(x_i; \mu_k, I) / Z$$

If we treat $r_{k,i}$ values as fixed, then, MLE objective becomes

$$\sum_i \sum_k r_{k,i} \left(\log \pi_k + \log \mathcal{N}(x_i; \mu_k, I)\right)$$

$\nabla_{\mu,\pi} LL_g(\mu_k, \pi_k) = 0$, solve for μ_k, π_k

$$\mu_k = \frac{\sum_i r_{k,i}\, x_i}{\sum_i r_{k,i}}, \qquad \pi_k = \frac{\sum_i r_{k,i}}{N}$$

($r_{k,i}$: weight that x_i commits to μ_k)

This motivates EM algorithm.

EM iteratively finds a lower bound of the log likelihood and optimizes it.

Initialise: $\mu_k^{curr} = \mu_k^{init}$, $\pi_k^{curr} = \pi_k^{init}$ $\forall k \in \{1, 2, \cdots, n_y\}$

E-step: Compute posterior probs.

$$\gamma_{k,i} = \frac{\pi_k^{curr} \, \mathcal{N}(x_i; \mu_k^{curr}, I)}{\sum_j \pi_j^{curr} \, \mathcal{N}(x_i; \mu_j^{curr}, I)}$$

M-step: Estimate μ_k, π_k

$$\mu_k^{new} = \frac{\sum_i \gamma_{k,i} \, x_i}{\sum_i \gamma_{k,i}}, \quad \pi_k^{new} = \frac{\sum_i \gamma_{k,i}}{N}$$

Check if the log-likelihood value is within a threshold of the previous log-likelihood

If yes, converged → terminate

else $\mu_k^{curr} = \mu_k^{new}$, $\pi_k^{curr} = \pi_k^{new}$ repeat E step and M step

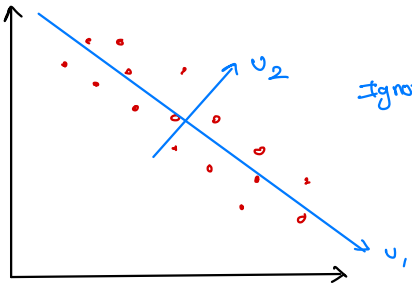
Dimensionality Reduction

Most popular dimension reduction technique is principal component analysis (PCA)

Consider a point $x \in \mathbb{R}^d$. Aim of PCA is to find a low dimensional projection of x, i.e.

$Z = U^T X$, $U \in \mathbb{R}^{d \times k}$ where $k < d$.

How do we estimate U?



Ignore u_2 ← loss of data but not very bad.

Minimize reconstruction error

Encode: $Z = U^T X$

Decode: $\hat{x} = UZ = UU^T X$

Reconstruction Objective fn

$$\min_{U^TU = I} \sum_i \|x_i - UU^T x_i\|^2 \quad -(1)$$

Objective 2 : Maximise the variance of the projected data

Mean-center the data, the resulting matrix is X. ($\mathbb{E}[X] = 0$)

$$\max_{U^T U = I} \mathbb{E}\left[\|U^T X\|^2\right]$$

For PCA, both objectives are equivalent

$$x = UU^T x + (I - UU^T)x$$

$$\|x\|^2 = \|UU^T x\|^2 + \|x - UU^T x\|^2 \quad \text{(cross-term = 0)}$$

$$\mathbb{E}[\|x\|^2] = \mathbb{E}[\|UU^T x\|^2] + \mathbb{E}[\|x - UU^T x\|^2]$$

{ U is a rotation, so $\mathbb{E}[\|Ux\|] = \mathbb{E}[\|y\|]$ }

$\mathbb{E}[\|x\|^2]$ fixed.

$$\mathbb{E}[\|x\|^2] = \mathbb{E}[\|U^T x\|^2] + \mathbb{E}[\|x - UU^T x\|^2]$$

Hence, both objectives are equivalent

Finding U_1, consider objective 2.

Assume, we are projecting down to $k = 1$.

$$\max_{\|u\|=1} \frac{1}{n} \sum_i (u^T x_i)^2$$

$$= \max_{\|u\|=1} \frac{1}{n}\|u^T X\|^2 = \max_{\|u\|=1} u^T \underbrace{\left(\frac{1}{n} XX^T\right)}_{\text{covariance matrix } (S)} u = \text{largest eigenvalue of } S.$$

Why ?

$$\max_{\|u\|^2 \le 1} u^T S u$$

$$L(u, \lambda) = u^T S u + \lambda(1 - u^T u)$$

$$\nabla_u L = 0 \Rightarrow 2Su - 2\lambda u = 0$$

$$\Rightarrow Su = \lambda u.$$

u is an eigenvector of S, with eigenvalue λ.

$$\nabla_\lambda L = 0 \Rightarrow u^T u = 1.$$

$u^T S u = \lambda$, so pick u with largest eigenvalue

PCA :

Compute the covariance matrix S, and pick k eigenvectors of S with the highest eigenvalues

17/10/23

Lecture

1. Tensor puzzles (coding practice)

Bagging

Recall the bias variance decomposition for regression

$$\mathbb{E}\left[(y - h(x))^2\right] = \underbrace{\left(\bar{h}(x) - f(x)\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}\left[(h(x) - \bar{h}(x))^2\right]}_{\text{variance}} + \sigma^2$$

($f(x)$: true value)

Consider constructing m independent datasets $D_1, D_2, \ldots, D_m$ all sampled from distribution P.

We can train a predictor for each of the n datasets $h(x; D_i)$

Average predictor $\frac{1}{m}\sum_{i=1}^{m} h(x; D_i)$

What happens to the bias and variance with using an average predictor.

$$\mathbb{E}_{D_1,\ldots,D_m}\left[\frac{1}{m}\sum_{i=1}^{m} h(x; D_i)\right] = \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}_{D_1,\ldots,D_m} h(x; D_i) = \mathbb{E}_D\, h(x; D)$$

Variance

$$\mathrm{Var}_{D_1,\ldots,D_m}\left[\frac{1}{m}\sum_{i=1}^{m} h(x; D_i)\right] = \frac{1}{m^2}\sum_i \mathrm{Var}_{D_i}\left[h(x; D_i)\right] = \frac{1}{m}\mathrm{Var}_D\left(h(x, D)\right)$$

Catch: We don't have m datasets, we only have one.

Solution: Bootstrap (Create multiple datasets from D using sampling with replacement)

Motivation for Bagging → Bootstrap Aggregation

Bagging Algorithm

- Consider a dataset D with n training examples
- Create m different datasets, each of size n by sampling from D with replacement
- Average the predictions (for regression) and consider majority vote across predictions from models trained on each of the m datasets.

Note: Since the m datasets are not independent, we don't get the $1/m\,\sigma^2$ variance reduction. Can show that if the sampled predictions have a variance σ^2, and a covariance of ρ^2 then,

$$\mathrm{Var}\left[\frac{1}{m}\sum_i h(x; D_i)\right] = \frac{1}{m}(1-\rho)\sigma^2 + \rho\sigma^2$$

Each of these m datasets are referred to as Bootstrap samples.

A bootstrap sample contains roughly 63.2% of the observed training points in D.

Why? If we sample from a uniform distribution the prob. of a single dataset example not being drawn as a bootstrap sample datapoint is

$$P(\text{not drawn}) = \left(1 - \frac{1}{n}\right)^n$$

$$\text{As } n \to \infty \quad = 1/e \approx 0.368.$$

The remaining instances in D that do not appear in a bootstrap sample are called "out of bag" samples [OOB]

OOB samples can be used to derive a bag estimate of test error

Process :

- For each i^{th} training example, predict the labels using all models trained using bootstrap sample for which i is OOB
- Aggregate the predicted values and compute the error for the i^{th} training eg.
- Average the error across all n training examples

Random Forest

Decision trees + bagging + Tweak further to de-correlate trees

- Create bootstrap samples
- Train a decision tree for each bootstrap sample
- Tweak : For every split in every DT, only consider a subset of features.

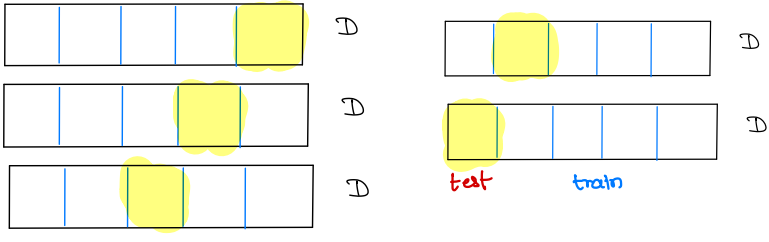
If there are d features, we lose a random subset of size k (where typically, $k \approx \sqrt{d}$) at each split for every DT.

Cross-Validation

Technique used for model evaluation / selection.

(k-fold) ← partition

k = 5



Leave-one-out cross validation (k=n)

Typically k = 3, 5, 10.

Note: Every example shows up in both training, test at some point