

Lec 00 Introduction

Prerequisites = Linear Algebra,
Calculus,
Probability,
Algorithm

History : Dartmouth Summer Research Project (1955)

Categories = Supervised,
Unsupervised,
Reinforcement,

Semi-supervised / Self-supervised / active learning.

Focus of ML Research

- Representation
how to encode the data?
- Generalization
how well can we do on unseen data?
- Interpretation
how to explain the findings?
- Complexity
how much time and space?
- Efficiency
how many samples?
- Privacy
how to respect data privacy?
- Robustness
how to degrade gracefully under (malicious) error?
- Fairness
how to enforce algorithmic equality?

Lectures :

Classic = Lec 00 ~ 06 Neural Nets = Lec 07 ~ 11 Generative Models = Lec 14 ~ 18 Nascent = Lec 19 ~ 23
12 ~ 13

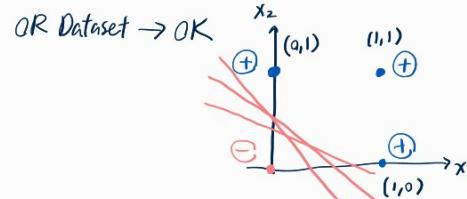
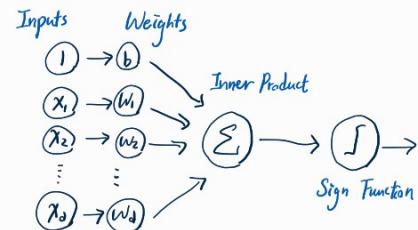
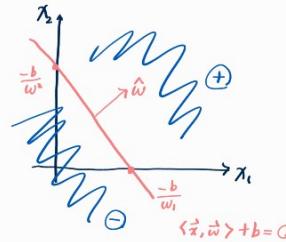
Lec 01 Perceptron

Batch Learning vs Online Learning

- Statistical assumption on training set vs test set
- Streaming data
- interested in making as few mistakes as possible

Linear Threshold Function (LTF)

$$\hat{y} = \text{sign}(\langle \vec{x}, \vec{w} \rangle + b) = \begin{cases} 1, & \hat{y} > 0 \\ -1, & \hat{y} < 0 \\ ?, & \hat{y} = 0 \end{cases}$$



Perceptron Algorithm (1958 by Rosenblatt)

Input: Dataset = $\{(x_i, y_i) \in \mathbb{R}^d \times \{\pm 1\}, i=1, \dots, n\}$

Initialization: $\vec{w} \in \mathbb{R}^d, b \in \mathbb{R}$

Threshold: $\xi > 0$ (typically $\xi = 0$)

Output: Approximate Solution: \vec{w}, b

Algorithm: for $t=1, 2, \dots$ do

let $k \in \{1, \dots, n\}$ be an index
if $y_k (\langle \vec{x}_k, \vec{w} \rangle + b) \leq \xi$ then
 $\vec{w} \leftarrow \vec{w} + y_k \vec{x}_k$
 $b \leftarrow b + y_k$

key update: if it ain't broke, don't fix it

Solves Optimization Problem = Find $\vec{w} \in \mathbb{R}^d, b \in \mathbb{R}$, st.

$$\forall i, y_i (\langle \vec{x}_i, \vec{w} \rangle + b) > 0$$

Key Insight: When a mistake happens,

$$y (\langle \vec{x}, \vec{w}_{\text{old}} \rangle + b_{\text{old}}) = y (\langle \vec{x}, \vec{w}_{\text{old}} + y \vec{x} \rangle + b_{\text{old}} + y)$$

$$= y (\langle \vec{x}, \vec{w}_{\text{old}} \rangle + b_{\text{old}}) + \|\vec{x}\|_2^2 + 1$$

Notation Simplification: $y[\langle \vec{x}, \vec{w} \rangle + b] = \underbrace{\langle y, \vec{x} \rangle}_{\text{padding}} + \underbrace{\langle \vec{w}, \vec{x} \rangle}_{\vec{w}} + b$

Problem becomes: Find $\hat{w} \in \mathbb{R}^p$ st. $A^T \hat{w} > \vec{0}$, where $A = [\vec{a}_1, \dots, \vec{a}_n] \in \mathbb{R}^{p \times n}$ Linear Inequalities

Convergence Theorem

If \exists a strictly separating hyperplane, the Perceptron iterate converges to some \hat{w} .
If each training data is selected infinitely often, then $\forall i, \langle y_i \vec{x}_i, \hat{w} \rangle > \delta$.

Boundness Theorem

Otherwise if not strictly separating, (\vec{w}, b) is always bounded and the perceptron cycles.

- Stops when
 - Max Iteration
 - Max Runtime
 - Training Error stagnant
 - Validation Error stagnant
 - Weight changes < tolerance (when using a step size)

Perception vs Support Vector Machine vs Soft-margin Support Vector Machine
 * Unique Solution * Beyond Separability

Lec 02 Linear Regression

Regression: Given dataset $\{(x_i, y_i)\}$, find $f: X \rightarrow Y$ st. $f(x_i) \approx y_i$

Statistical Learning

i.i.d = Independently and Identically drawn samples from a distribution P
 $(x_i, y_i) \sim P$ and $(X, Y) \sim P$

Assumption: Law of Large numbers in Expectation

Least Squares Regression = $\min_{X \rightarrow Y} E \|f(X) - Y\|_2^2$ Approx. Error
 → Sampling (Training) $\frac{1}{n} \sum_{i=1}^n \|f(X_i) - Y_i\|_2^2$ Estimation Error

bias² noise variance
 Aim: from minimize bias does not depend on f
inherit measure of the difficulty of the problem.

Padding + Matrix Form = $\min_{\hat{w} \in \mathbb{R}^{(p+1)}} \frac{1}{n} \|\hat{w} \hat{X} - \hat{Y}\|_F^2$ Tranenius Norm

Solving: $\hat{w} = \hat{Y} \hat{X}^T (\hat{X} \hat{X}^T)^{-1}$ depends on invertibility
 Apply Iterative Gradient Algorithms!!!!

\hat{X} can be ill-conditioning e.g. $\hat{X} = \begin{bmatrix} 0 & \varepsilon \\ 1 & 1 \end{bmatrix} \rightarrow \hat{w} = \hat{y} \hat{X}^{-1} = \left[-\frac{2}{\varepsilon}, 1 \right]$ Slight Perturbation leads to chaotic behaviour

Ridge Regression = $\min_{\hat{w}} \frac{1}{n} \|\hat{w} \hat{X} - \hat{Y}\|_F^2 + \lambda \|\hat{w}\|_F^2$ Regularization \Leftrightarrow Constraints

$$\min_{\|\hat{w}\|_F \leq R} \frac{1}{n} \|\hat{w} \hat{X} - \hat{Y}\|_F^2$$

Data Augmentation = $\frac{1}{n} \|\hat{w} [\hat{X} \quad \sqrt{\lambda} I]\] - [\hat{Y} \quad \hat{0}]\|_F^2$ restricts output to be $\frac{1}{\sqrt{\lambda}}$ proportion to input
 = regularization shrinks $\hat{w} \rightarrow \hat{0}$ when $\lambda \rightarrow \infty$
 arguments p data points $x_j = j\sqrt{\lambda} e_j$



Lasso Regression = $\min_{\hat{w}} \frac{1}{n} \|\hat{w} \hat{X} - \hat{Y}\|_F^2 + \lambda \|\hat{w}\|_1$ Regularization \Leftrightarrow Constraints

$$\min_{\|\hat{w}\|_1 \leq R} \frac{1}{n} \|\hat{w} \hat{X} - \hat{Y}\|_F^2$$

Task Regularization : $\min_{\hat{w}_t} \frac{1}{n} \|\hat{w}_t \hat{X} - \hat{Y}_t\|_F^2 + \lambda \|\hat{w}_t\|_2^2 \quad \forall t=1, \dots, T$

Assumption: Tasks are independent and can be solved separately
 ⇒ Cross Validation • k-fold validation

$$\text{perf}(\lambda) = \text{perf}_1 + \text{perf}_2 + \dots + \text{perf}_K$$

$$\lambda^* = \underset{\lambda}{\operatorname{argmax}} \text{perf}(\lambda)$$



Lec 03 Logistic Regression

Motivation: turn the output of the classifier e.g. $\hat{y} = \text{sign}(\hat{w}^T \hat{x} + b)$

Parametrization of Binary Classification Problem

~~Posterior probability~~ Let $P(\vec{x}; \vec{w})$ approx. $\Pr(Y=1 | X=\vec{x})$ where labels $\vec{y} \in \{\pm 1\}$
 where $p(\vec{x}; \vec{w}) \in [0, 1]$



$$\text{Bernoulli Model: } \Pr(Y=y | X=\vec{x}) = p(\vec{x}; \vec{w})^{\frac{1+y}{2}} (1-p(\vec{x}; \vec{w}))^{\frac{1-y}{2}}$$

$$\text{Conditional Likelihood: } \Pr(Y=y_1, \dots, Y_n=y_n | X_1=\vec{x}_1, \dots, X_n=\vec{x}_n)$$

$$\text{Maximize this Conditional likelihood} = \prod_{i=1}^n p(\vec{x}_i; \vec{w})^{\frac{1+y_i}{2}} (1-p(\vec{x}_i; \vec{w}))^{\frac{1-y_i}{2}}$$

w.r.t. \vec{w} ! → figure out \vec{w} → make probability estimates with this \vec{w} and new test \vec{x} .

Logistic Regression (1958)

Likelihood from Bernoulli Model + Sigmoid Function as probability estimates

Picking $p(\vec{x}; \vec{w})$:

$p(\vec{x})$? diff for every data point, too flexible

$p(\vec{w})$? all same, too inflexible

$\langle \vec{x}, \vec{w} \rangle + b$? does not map to $[0, 1]$; $\langle \vec{x}, \vec{w} \rangle + b \rightarrow \mathbb{R}$.

Logit Transformation

$$\log \frac{p(\vec{x}; \vec{w})}{1-p(\vec{x}; \vec{w})} \leftarrow \text{odds ratio; map to } \mathbb{R} \text{ same as } \langle \vec{x}, \vec{w} \rangle + b.$$

$$\text{equivalently } p(\vec{x}; \vec{w}) = \frac{\exp(\langle \vec{x}, \vec{w} \rangle + b)}{1 + \exp(\langle \vec{x}, \vec{w} \rangle + b)} = \text{Sigmoid}(\langle \vec{x}, \vec{w} \rangle + b) \in [0, 1]$$

Logistic Loss (after plugging into the Likelihood equation)

$$\min_{\vec{w}} \sum_{i=1}^n \log(1 + \exp(-y_i \langle \vec{x}_i, \vec{w} \rangle))$$

Cross-Entropy; objective $KL\left(\frac{1+y}{2} \parallel \hat{p}\right)$

Does not enjoy Closed-Form Solution

Iteratively converge to a solution

① Gradient Descent = apply Chain Rule (compute gradient)

② Newton's Algorithm (compute gradient + Hessian) much faster, but computing + storing Hessian is expensive.
 ~ Iterative re-weighted linear regression (least square problem)

Multiclass Logistic Regression

- Multinomial Logit $\hat{p}_c = f(\vec{W} \vec{x}_c) = \text{softmax}(\vec{W} \vec{x}_c)$, softmax: $\mathbb{R}^C \rightarrow \Delta_{C-1}$,
- Softmax Regression

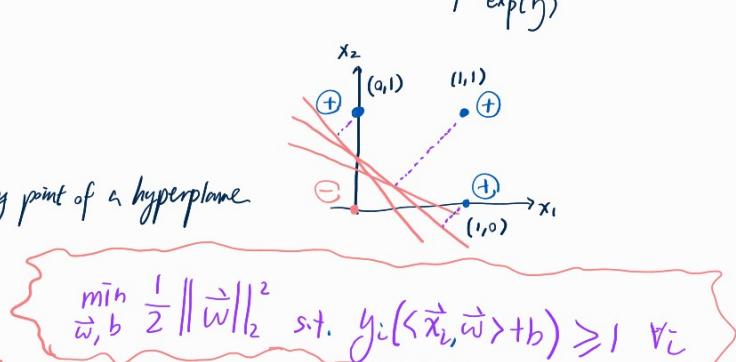
$$\eta \mapsto \frac{\exp(\eta)}{\sum_j \exp(\eta)}$$

Lec 04 Support Vector Machines

Motivation: Maximizing Margin, i.e.

Maximizing the minimum distance to every point of a hyperplane

$$\max_{\vec{w}, b} \min_i \frac{|y_i (\langle \vec{x}_i, \vec{w} \rangle + b)|}{\|\vec{w}\|_2}$$



Quadratic Programming (Perceptron = Linear Programming)
 Unique Solution (Perceptron: infinite solutions)

$$\min_{\vec{w}, b} \frac{1}{2} \|\vec{w}\|_2^2 \text{ s.t. } y_i (\langle \vec{x}_i, \vec{w} \rangle + b) \geq 1 \quad \forall i$$

Primal form

Support Vectors: points lie on the hyperplane (decisive)

Convex Set: $C \subseteq \mathbb{R}^d$ is convex iff $\forall \vec{x}, \vec{y} \in C$ and $\forall \alpha \in [0, 1]$,

$$(1-\alpha)\vec{x} + \alpha\vec{y} \in C$$

i.e. the line segments connecting any two points in C remains in C .

Example: \emptyset, \mathbb{R}^d (vector space)

Convex Hull: The convex hull $\text{conv}(A)$ of a set A is the intersection of all convex supersets of A , i.e.

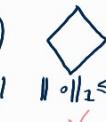
$$\text{conv}(A) := \bigcap_{\text{convex } C \supseteq A} C$$

i.e. the smallest convex superset.

Example: Unit balls of norms

$$B_p := \{ \|\vec{x}\|_p \leq 1 \}$$

is convex iff $p \geq 1$



$$\text{conv}(B_{\frac{1}{2}}) = \text{conv}(B_1)$$

Convex Combination: of a finite set of points $\vec{x}_1, \dots, \vec{x}_n$ as

$$\vec{x} = \sum_{i=1}^n \alpha_i \vec{x}_i \quad \text{where } \alpha_i \geq 0, \quad \vec{\alpha}^T \vec{\alpha} = 1$$

For $A \subseteq \mathbb{R}^d$

$$\text{conv}(A) = \left\{ \vec{x} = \sum_{i=1}^n \alpha_i \vec{x}_i : \begin{array}{l} n \in \mathbb{N} \\ \vec{\alpha} \in \Delta_{n-1} \\ \vec{x}_i \in A \\ \alpha_i \geq 0, \quad \sum_i \alpha_i = 1 \end{array} \right\}$$

Convex hull = the set of all convex combinations of points in A

Dual View of SVM (1965)

A dataset D is (strictly) linearly separable

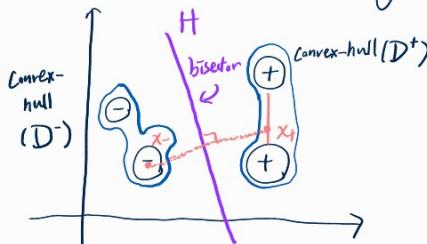
$$\text{iff } \text{convex-hull}(D^+) \cap \text{convex-hull}(D^-) = \emptyset$$

$$\text{where } D^\pm := \{ \vec{x}_i \in D : y_i = \pm 1 \}$$

Let H : a separating hyperplane that passes the middle point $\frac{1}{2}(\vec{x}_+ + \vec{x}_-)$ as a bisector

where $\vec{x}_+ \in \text{convex-hull}(D^+)$, $\vec{x}_- \in \text{convex-hull}(D^-)$ so that

$\text{dist}(\vec{x}_+, \vec{x}_-)$ achieves the minimum distance among all pairs from the two convex hulls.



Maximize the minimum distance among all pairs from the two convex-hulls

$$\max_{\|\vec{w}\|=1, b} \text{dist}(D^+, H_{\vec{w}}) \wedge \text{dist}(D^-, H_{\vec{w}}) = \frac{1}{2} \text{dist}(\text{conv}(D^+), \text{conv}(D^-))$$

H : solution of SVM = i.e. maximize the min. distance to every training samples in D

Lec 05 Soft-Margin Support Vector Machine

Motivation: beyond separability

$$\min_{\vec{w}, b} \frac{1}{2} \|\vec{w}\|_2^2 + C \cdot \sum_{i=1}^n (1 - y_i \vec{w}^T \vec{x}_i)^+$$

$$\text{s.t. } \hat{y}_i = \langle \vec{x}_i, \vec{w} \rangle + b \quad \forall i$$

$$w, b$$

margin maximization

hinge loss $(1-y\hat{y})^+$
 i -th training error, 0 if $y_i \hat{y}_i \geq 1$

Soft Penalty: the more deviate, the heavier the penalty

Update Rule:

$$\text{Gradient Descent} = \vec{w} \leftarrow \vec{w} - \eta \left[\frac{1}{n} \sum_{i=1}^n \ell'(y_i \hat{y}_i) y_i \vec{x}_i + \frac{\vec{w}}{\lambda} \right]$$

$$\text{Stochastic Gradient: } \vec{w} \leftarrow \vec{w} - \eta \left[\underset{\text{(random sample suffices)}}{\cancel{\sum}} \ell'(y_i \hat{y}_i) y_i \vec{x}_i + \frac{\vec{w}}{\lambda} \right]$$

derivative of loss function

$$\ell'_{\text{hinge}}(t) = \begin{cases} -1, & t < 1 \\ 0, & t \geq 1 \\ [1, 0], & t=1 \end{cases}$$

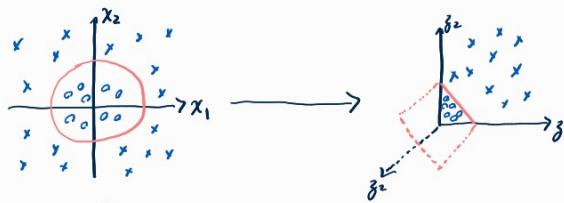
✓ Multi-class Vector Machines

✓ Support Vector Regression Machine

✓ Maximal Margin Clustering

Lec 06 Reproducing Kernels

Motivation: from non-linear to linear



Example: Quadratic Classifier: The Power of Lifting!

$$f(\vec{x}) = \langle \vec{x}, Q\vec{x} \rangle + \sqrt{2}\langle \vec{x}, \vec{p} \rangle + b$$

$$\text{predicts: } \hat{y} = \text{sign}(f(\vec{x}))$$

$$\text{Weights: } Q \in \mathbb{R}^{d \times d}$$

$$\vec{p} \in \mathbb{R}^d$$

$$b \in \mathbb{R}$$

Linear when $Q = \vec{0}$

$$f(\vec{x}) = \langle \phi(\vec{x}), \vec{w} \rangle \text{ where}$$

$$\text{Feature Map: } \phi(\vec{x}) = \begin{bmatrix} \vec{x} \vec{x}^T \\ \sqrt{2} \vec{x} \\ 1 \end{bmatrix} \quad \vec{w} = \begin{bmatrix} Q \\ \vec{p} \\ b \end{bmatrix}$$

$$\text{Weights: } \phi(\vec{x}) \in \mathbb{R}^{d \times d+1} \quad \vec{w} \in \mathbb{R}^{d \times d+1}$$

Non-linear in \vec{x} , linear in $\phi(\vec{x})$

Kernel Trick Lifting up to higher-dimensions - blows up?

$$\text{All needed is the inner product } \langle \phi(\vec{x}), \phi(\vec{z}) \rangle = (\langle \vec{x}, \vec{z} \rangle)^2 + 2\langle \vec{x}, \vec{z} \rangle + 1$$

$= (\langle \vec{x}, \vec{z} \rangle + 1)^2$ still computes in $O(d)$ time

(Reproducing) Kernels

$k: X \times X \rightarrow \mathbb{R}$ is a (reproducing) kernel if \exists some feature transformation $\phi: X \rightarrow \mathcal{H}$ Hilbert space

$$\text{s.t. } \langle \phi(\vec{x}), \phi(\vec{z}) \rangle = k(\vec{x}, \vec{z})$$

$$\text{Reproducing: } \langle f, k(\cdot, \vec{x}) \rangle = f(\vec{x}) \quad \forall f \in \mathcal{H}$$

Choose $\phi \rightarrow$ determines k

and $\langle k(\cdot, \vec{x}), k(\cdot, \vec{z}) \rangle = k(\vec{x}, \vec{z})$

determines some $\phi \leftarrow$ Choose k

decides a unique RKHS (reproducing kernels Hilbert space)

$$H_k := \{ \vec{x} \mapsto \langle \phi(\vec{x}), f \rangle : f \in \mathcal{H} \} \subseteq \mathbb{R}^X \quad k \leftrightarrow H_k : 1-1 \text{ correspondence}$$

Verify a Kernel

Thm: $k: X \times X \rightarrow \mathbb{R}$ is a kernel iff $\forall \vec{x}_1, \dots, \vec{x}_n \in X$,

$$\text{the kernel matrix } K_{ij} := (k(\vec{x}_i, \vec{x}_j)) \in \mathbb{S}_+^n \text{ symmetric and positive semi-definite (PSD)}$$

Some form of similarity measure

$$K_{ij} = k_{j_i}$$

$$\forall \alpha \in \mathbb{R}^n \quad \langle \alpha, K\alpha \rangle = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j K_{ij} \geq 0$$

Examples:

Polynomial kernel: $k(\vec{x}, \vec{y}) = (\langle \vec{x}, \vec{y} \rangle + 1)^p$ Gaussian kernel: $k(\vec{x}, \vec{y}) = \exp\left(-\frac{\|\vec{x} - \vec{y}\|_2^2}{\sigma^2}\right)$ infinite-dimensional RKHSLaplace kernel: $k(\vec{x}, \vec{y}) = \exp\left(-\frac{\|\vec{x} - \vec{y}\|_2}{\sigma}\right)$ Browniian kernel: $k(s, t) := s \wedge t \quad \forall s, t \geq 0$

Universal Kernel

A kernel is universal if its RKHS is large enough to approximate any continuous function (over a compact domain X)

Kernel SVM, Kernel Logistic Regression

Dual form of SVM

$$\min_{C \geq 0, \alpha \geq 0} - \sum_i \alpha_i + \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle \vec{x}_i, \vec{x}_j \rangle \quad \text{s.t. } \alpha_i y_i = 0$$

\downarrow
 $k(\vec{x}_i, \vec{x}_j)$

no need to know ϕ

$$\text{Testing: } f(\vec{x}) := \langle \phi(\vec{x}), \vec{w} \rangle = \left\langle \phi(\vec{x}), \sum_i \alpha_i y_i \phi(\vec{x}_i) \right\rangle = \sum_i \alpha_i y_i k(\vec{x}, \vec{x}_i) \in \mathcal{H}_k$$

Knowing the dual variable α , training set $\{\vec{x}_i, y_i\}$, kernel k suffices!

New

Training: $O(n^2 d)$

Testing: $O(nd)$

Training: $O(nd)$

Testing: $O(d)$

Old

Trade-Off

✓ avoid explicit dependence on feature dimension (e.g. ∞)✗ n times slower in both training + testing

✗ need to store training set (in case of SVM)

Kernel Logistic Regression

$$\min_{\vec{w}} \frac{1}{n} \sum_{i=1}^n \log (1 + \exp(-y_i \langle \vec{x}_i, \vec{w} \rangle)) + \frac{\lambda}{2} \|\vec{w}\|_2^2$$

\downarrow
 $\langle \phi(\vec{x}_i), \vec{w} \rangle$

Representer Theorem

$$\vec{w}^* = \sum_{j=1}^n \alpha_j y_j \phi(\vec{x}_j) \quad \text{for some } \vec{\alpha} \in \mathbb{R}^n$$

Orthogonal Decomposition

$$\vec{w} = \vec{w}^\parallel + \vec{w}^\perp, \quad \vec{w}^\parallel \in \text{span} \{ y_i \phi(\vec{x}_i) \mid i \in \{1, \dots, n\} \}$$

Logistic Loss only depends on \vec{w}^\parallel !

Learning the Kernel

Motivation: learn a pos. combination of base kernels with coefficient β , learned simultaneously with \vec{w}

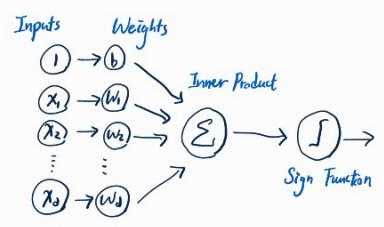
$$\min_{\vec{w}} \frac{1}{n} \sum_i l(\langle \phi(\vec{x}_i), \vec{w} \rangle) + \frac{\lambda}{2} \|\vec{w}\|_2^2$$

$$\Rightarrow \min_{\vec{w}, \beta} \frac{1}{n} \sum_i l\left(\left\langle \sum_p \beta_p \phi_p(\vec{x}_i), \vec{w} \right\rangle\right) + \frac{\lambda}{2} \|\vec{w}\|_2^2$$

$$\text{Representor Theorem } \vec{w} = \sum_j \alpha_j \oplus \beta_j \phi_p(\vec{x}_j)$$

Lec 08 Multi-Layer Perception

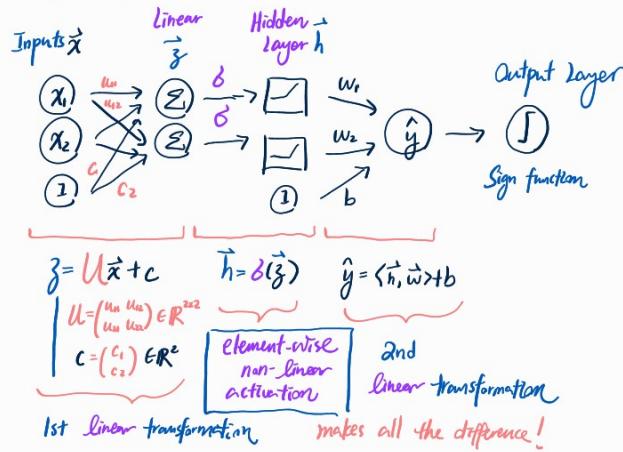
Recall: Perceptron



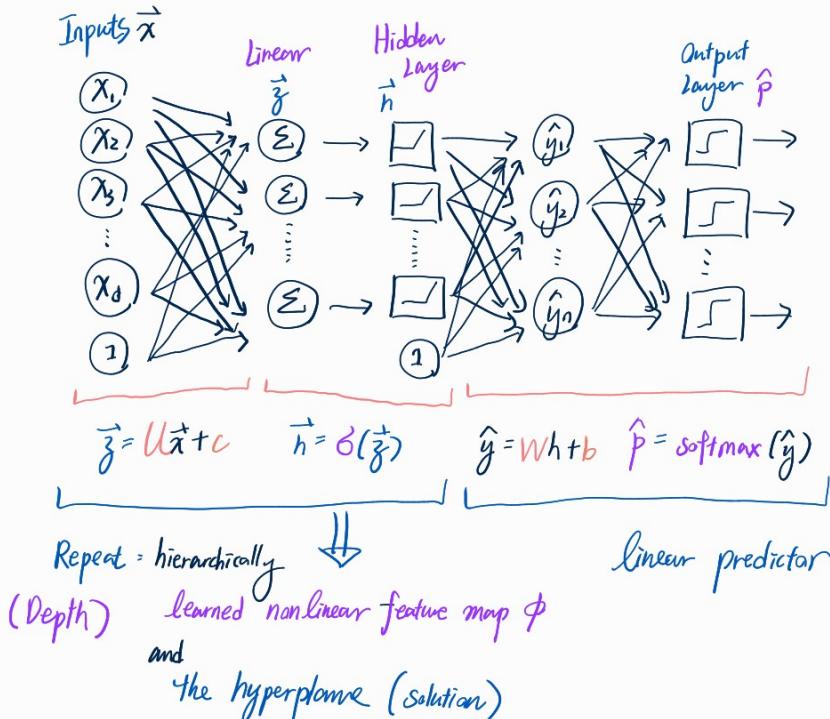
Algorithm: for $t=1, 2, \dots$ do
let $k \in \{1, \dots, n\}$ be an index
if $y_k \cdot (\langle \vec{x}_k, \vec{w} \rangle + b) \leq \delta$ then
 $\vec{w} \leftarrow \vec{w} + y_k \vec{x}_k$
 $b \leftarrow b + y_k$

Solves Optimization Problem: Find $\vec{w} \in \mathbb{R}^d, b \in \mathbb{R}$, st.
 $\forall i, y_i (\langle \vec{x}_i, \vec{w} \rangle + b) > 0$

Case: $\vec{x} \in \mathbb{R}^2$



Case: $X \in \mathbb{R}^d$, multi-class Feed-forward MLP



Algorithm (Test time)

$h_0 \leftarrow \vec{x}$
for $k=1, \dots, l$ do (feature map: layer by layer)
| $\vec{z}_k \leftarrow W_k h_{k-1} + b$
| $h_k \leftarrow \delta(\vec{z}_k)$
 $\hat{y} \leftarrow W_{l+1} h_l + b_{l+1}$
 $\hat{p} \leftarrow \text{softmax}(\hat{y})$

Activation Functions

Sigmoid (t) = $\frac{1}{1 + \exp(-t)}$

$\tanh(t) = 1 - 2 \text{Sigmoid}(2t)$

ReLU (t) = t_+

ELU (t) = $(t)_+ + (t)_- \cdot (\exp(t) - 1)$

MLP Training

Loss function ℓ : measure differences between \hat{p} and truth y
 e.g. squared loss $\|\hat{p} - y\|_2^2$, log-loss $-\log \hat{p}_y$

Training set $D = \{(\vec{x}_i, y_i) : i=1, \dots, n\}$ to train weights \vec{w}

Stochastic Gradient Descent (SGD)

Each iteration, a random, minibatch $B \subseteq \{1, \dots, n\}$ suffices

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \frac{1}{|B|} \sum_{i \in B} \nabla [\ell(f(\vec{x}_i; \vec{w}), y_i)]$$

Trade-off = variance \propto computation \downarrow

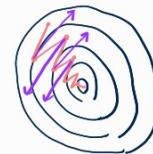
instead of $\frac{1}{n} \sum_{i=1}^n$ (full pass over the training set)

Momentum

Interpretation: gradient averaging

$$\vec{w}_{t+1} = \vec{w}_t - \eta_t \nabla g(\vec{w}_t) + \beta_t (\vec{w}_t - \vec{w}_{t-1})$$

gradient step momentum



Nesterov Momentum: averaging gradients looking ahead

$$\vec{w}_{t+1} = \vec{z}_t - \eta_t \nabla g(\vec{w}_t), \quad \vec{z}_{t+1} = \vec{w}_{t+1} + \beta_t (\vec{w}_{t+1} - \vec{w}_t)$$

looks ahead

Adjust Learning Rate

$$\vec{w}_{t+1} = \vec{w}_t - \eta_t \left(\frac{1}{S_t + \varepsilon} \right) \odot \vec{v}_t$$

Adagrad (Adaptive Gradient)

$$S_t = S_{t-1} + \nabla g(\vec{w}_t) \odot \nabla g(\vec{w}_t)$$

$$\vec{v}_t$$

RMSprop (Root Mean Square Propagation)

$$(1 - \lambda_t) S_{t-1} + \lambda_t \nabla g(\vec{w}_t) \odot \nabla g(\vec{w}_t)$$

$$\nabla g(\vec{w}_t)$$

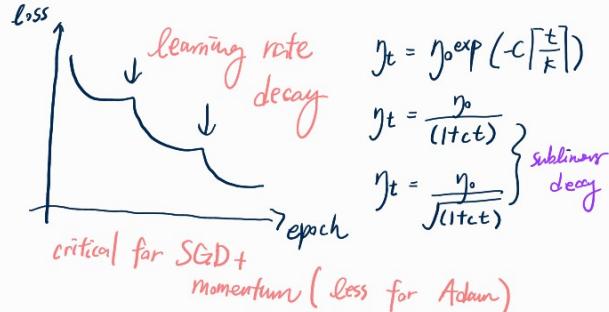
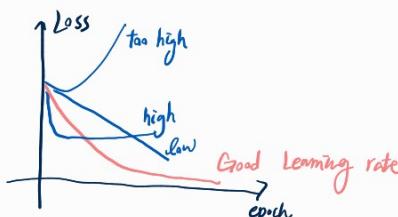
Adam (Adaptive Moment Estimation)

$$(1 - \lambda_t) S_{t-1} + \lambda_t \nabla g(\vec{w}_t) \odot \nabla g(\vec{w}_t)$$

$$\nabla g(\vec{w}_t)$$

$$(1 - \lambda_t) \vec{v}_{t-1} + \beta_t \nabla g(\vec{w}_t)$$

$$\vec{v}_t$$



Computational Graph

DAG = Direct Acyclic Graph

Forward vs Backward Differentiation

Universal Approximation

Let $\delta: \mathbb{R} \rightarrow \mathbb{R}$ be Riemann Integrable on any bounded interval.

Then, two-layer NN with activation δ is "dense" in $C(\mathbb{R}^d)$

iff δ is not a polynomial a.e. (almost everywhere)

Dropout

For each training minibatch, keep each hidden unit with probability γ
Use full network for testing less likely to overfit training data

Batch Normalization

Input: \vec{x} over a mini-batch = $B = \{x_1, \dots, x_m\}$

Parameters to be learnt = γ, β

Output = $\{y_i = BN_{\gamma, \beta}(x_i)\}$

Mini-batch Mean: $\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$

Variance: $\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$

Normalize: $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$

Scale and Shift: $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i)$

Lec 09

Convolutional Neural Network

Motivation: Weights sharing with a restricted field by filters



Typical layers: $\underbrace{\text{conv} \rightarrow \text{relu}}_{\text{repeat}} \rightarrow \underbrace{\text{conv} \rightarrow \text{relu}}_{\text{repeat}} \rightarrow \underbrace{\text{pool}}_{\text{input}} \rightarrow \underbrace{\text{FC} \rightarrow \text{relu}}_{\text{repeat}} \rightarrow \text{FC}$

Hierarchical Feature Representation input → low level features → mid level features → high level features → linearly separable classifier

Controlling the Convolution

- Filter size: $w \times h \times d$ (depth same as input by default)
- Number of Filters: weights not shared between filters
- Stride: number of pixels to move the filter each time
- Padding: add zero (or others) around boundary of input

Pooling: down-sample input size to reduce computation and memory
on each slice separately (output depth = input depth)
e.g. max-pool, average (l_p -)norm pool
size, stride: as in convolution, no parameter
no padding (typically)

Examples

LeNet AlexNet VGGNet GoogleNet (no FC, deeper but efficient)

Residual Block:

$h^{(I)} \xrightarrow{\text{Conv} \rightarrow \text{ReLU} \rightarrow \text{Conv} \rightarrow \text{ReLU}} h^{(I+2)}$
a residual block

Short-cut connection that allows "skipping" one or more layers
Allow more direct backpropagation of the gradient

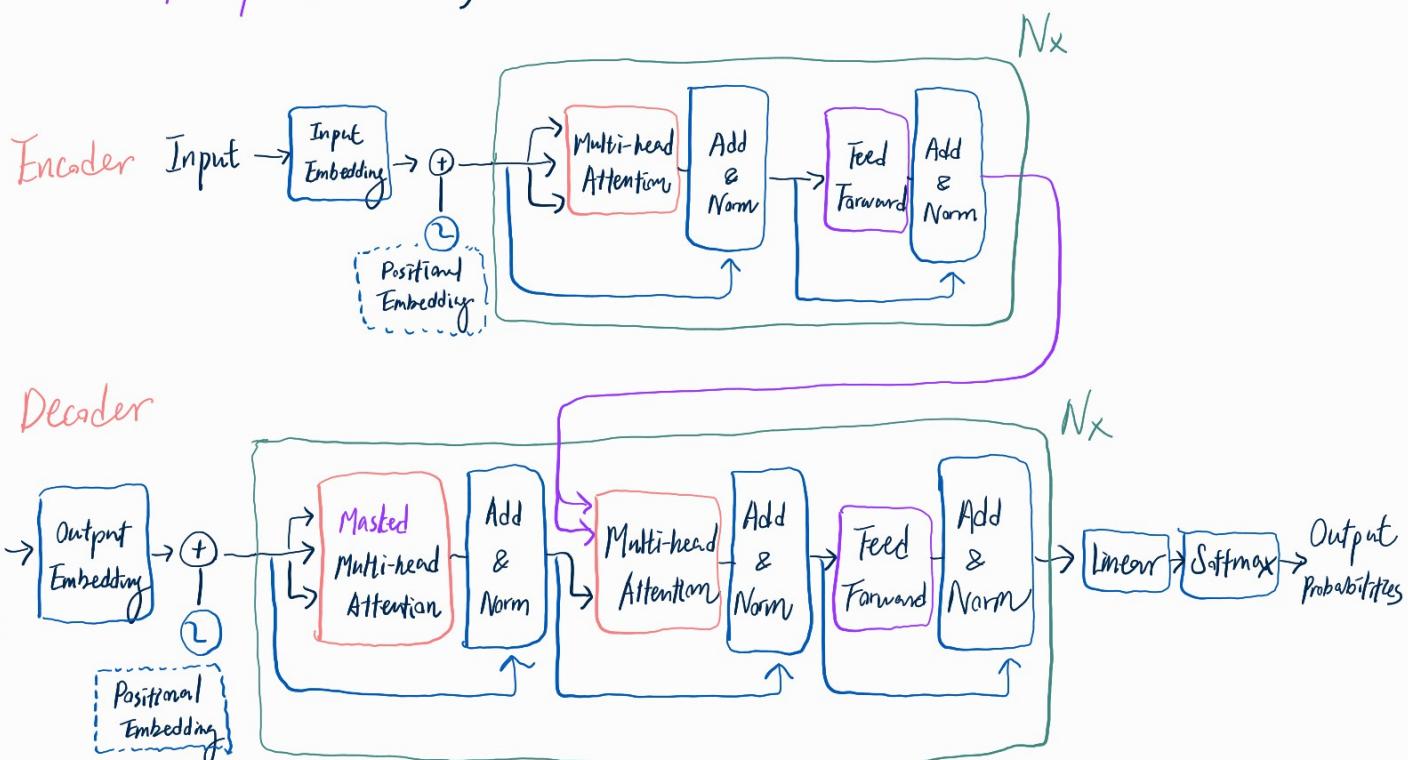
Examples:
ResNet

Lec 10 Attention

Motivation:

- | RNN: Recurrent Neural Network
- | → sequential / in Nature (w.r.t. input tokens)
- | use a hierarchical, parallelizable attention mechanism
- | to trade the sequential part in RNNs with network depth

Transformer (2017)



$$\text{Input Sequences } \begin{pmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_m \end{pmatrix} \quad \text{Output Sequence } \begin{pmatrix} \vec{y}_1 \\ \vec{y}_2 \\ \vdots \\ \vec{y}_e \end{pmatrix} \quad \vec{x}_m, \vec{y}_e \in \mathbb{R}^P \text{ one-hot}$$

P : dictionary size

$$\text{Embedding} = XW^e \text{ and } TW^e, \quad W^e \in \mathbb{R}^{P \times d}, \quad d = 512$$

$$\left(\begin{array}{c} \vec{x}_1 \\ \vdots \\ \vec{x}_m \end{array} \right) \cdot P \left\{ \left(\vec{w}_1 \vec{w}_2 \dots \vec{w}_d \right) \right\}$$

Learn a distributed representation of the input tokens
used to decode the output tokens

$$(\text{Additive}) \text{ Positional Encoding} = W^P \in \mathbb{R}^{\text{len} \times d}$$

$$\left(\begin{array}{c} \vec{0} \\ \vdots \\ \vec{0} \end{array} \right)$$

Encode the order of the tokens
(t : each fixed position)

$$W_{2i}^P = \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right) \quad W_{2i+1}^P = \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right)$$

$$i = 0, \dots, \frac{d}{2} + 1$$

The chosen periodic functions:
handle test sequence

Attention (2015)

Motivation: Given a Query vector $\vec{q} \in \mathbb{R}^{dk}$
and a database of m key-value pairs (K, V) :
 $K = [\vec{k}_1, \dots, \vec{k}_m]^T \in \mathbb{R}^{m \times d}$, $V = [\vec{v}_1, \dots, \vec{v}_m]^T \in \mathbb{R}^{m \times d}$

Guess the value of the Query:

$$\text{attention}(\vec{q}; K, V) = \sum_{i=1}^m \pi_i \cdot \vec{v}_i = \vec{\pi}^T V$$

context combination of
the values in the database

Solution of $\vec{\pi}_i$ (coefficient vector):

$$\underset{\vec{\pi} \in \Delta_m}{\operatorname{argmin}} \sum_{i=1}^m \pi_i \cdot \boxed{\text{dist}_k(\vec{q}, \vec{k}_i)} + \lambda \cdot \boxed{\pi_i \log \pi_i}$$

dissimilarity between
the query \vec{q} and the key \vec{k}_i

entropy regularizer

$$\Rightarrow \vec{\pi} = \text{softmax}\left(-\frac{\text{dist}_k(\vec{q}, K)}{\lambda}\right) \quad \pi_i = \frac{\exp\left(-\frac{\text{dist}_k(\vec{q}, \vec{k}_i)}{\lambda}\right)}{\sum_{j=1}^m \exp\left(-\frac{\text{dist}_k(\vec{q}, \vec{k}_j)}{\lambda}\right)}$$

Popular choice of dissimilarity function:

- $\text{dist}_k(\vec{q}, \vec{k}) = -\vec{q}^T \vec{k}$
 - parameterize as a Feed-Forward Network
 $\text{dist}_k(\vec{q}, \vec{k}; \vec{w}) \quad \vec{w} \text{ learns from the data}$
- ✓ if $\|\vec{q}\|_2 = \|\vec{k}\|_2 = 1$, $\vec{q}^T \vec{k}$ product - angular distance
 ✓ Efficient: implement $\vec{Q} \vec{K}^T$ in parallel
 ✓ set $\lambda = \sqrt{d}$ $\rightarrow \vec{q} \parallel \vec{k} \sim N(0, I_d)$
 $\text{Var}\left(\frac{\vec{q}^T \vec{k}}{\sqrt{d}}\right) = 1$

With $\vec{\pi}$, solve Weighted Regression problem for the value of the Query:

$$\underset{\vec{a}}{\operatorname{argmin}} \sum_{i=1}^m \pi_i \cdot \text{dist}_V(\vec{a}, \vec{v}_i)$$

Set $\text{dist}_V(\vec{a}, \vec{v}_i) = \|\vec{a} - \vec{v}_i\|_2^2 \rightarrow \text{verify this}$

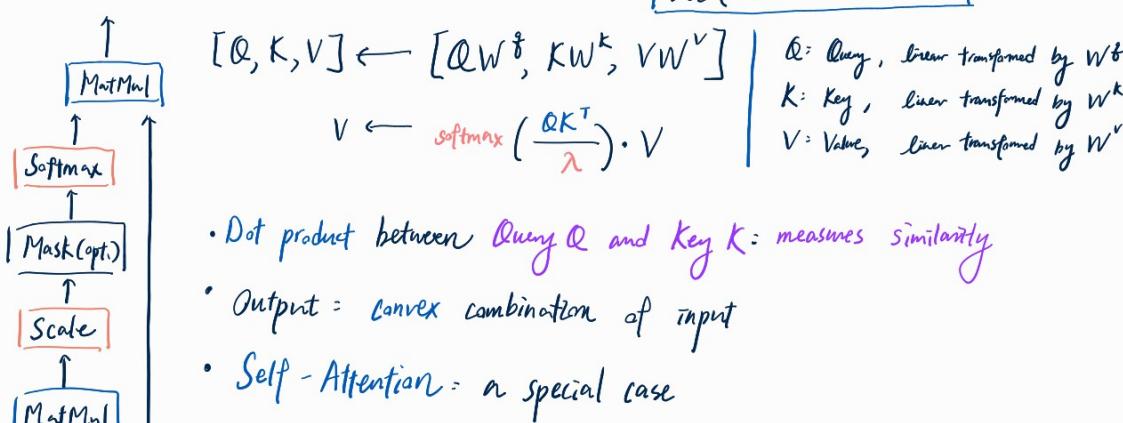
Self-Attention

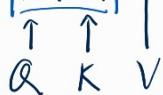
$$V \leftarrow VW^V \\ V \leftarrow \text{softmax}\left(\frac{VV^T}{\lambda}\right) \cdot V, \text{ e.g. } \lambda = \sqrt{d}$$

- Highly parallelizable matrix product, replace recurrence
- Dot product $\langle \vec{v}_i, \vec{v}_j \rangle$ measures similarity:
more similar, more contribution
- Each output = a convex combination of all inputs
- Softmax is dense: every output attends to every input

(General) Scaled Dot-Product Attention

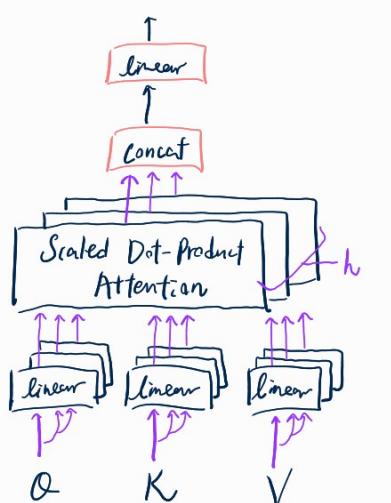
$$\text{att}(QW^Q, KW^K, VW^V)$$





$$[Q, K, V] \leftarrow [VW^T, VW^T, VW^T]$$

Multi-head Attention



$h = 8 \text{ or } 12$
(keep $d = 512$)

for $i=1, \dots, h$ do
 $V_i = \text{att}(QW_i^T, KW_i^T, VW_i^T)$
 // linear transformation
 $[Q, K, V] \leftarrow [QW_i^T, KW_i^T, VW_i^T]$
 // linear combination
 $V_i \leftarrow \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \cdot V$
 // concatenation & linear terms
 $V \leftarrow [V_1, \dots, V_h]W$

$W_i^T, W_i^k \in \mathbb{R}^{d_k \times d}$
 $W_i^v \in \mathbb{R}^{d_v \times d}$
 $W \in \mathbb{R}^{(h \cdot d_v) \times d}$
 Set $d_k = d_v = \frac{d}{h}$
 → num of parameters
 in h -head attention
 = single-head attention

⇒ Also facilitate the implementation of Residual Connections

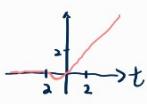
Position-wise Feed-Forward

$$\text{FFN}(\hat{V}) = \sigma(\hat{V}\hat{W}_2)\hat{W}_2 \longrightarrow \text{Two-layer Network}$$

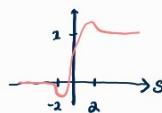
$\hat{V} \in \mathbb{R}^{len \times d}$
 → values at some layer, arranged row-wise
 $\hat{W}_1 \in \mathbb{R}^{d \times 4d}$
 $\hat{W}_2 \in \mathbb{R}^{4d \times d}$ } weights, shared among different positions,
 change from layer to layer

$\sigma(t) := \text{GELU}$ (Gaussian Error Linear Unit)

$$y(t) = t \cdot \sigma(t)$$

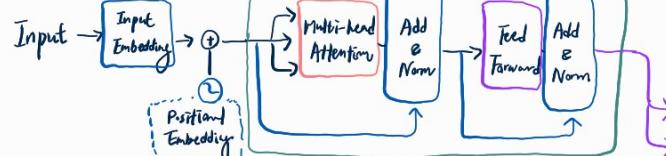


$$y'(t) = t\phi(t) + \sigma(t)$$



Encoder: Understanding Attention

Encoder



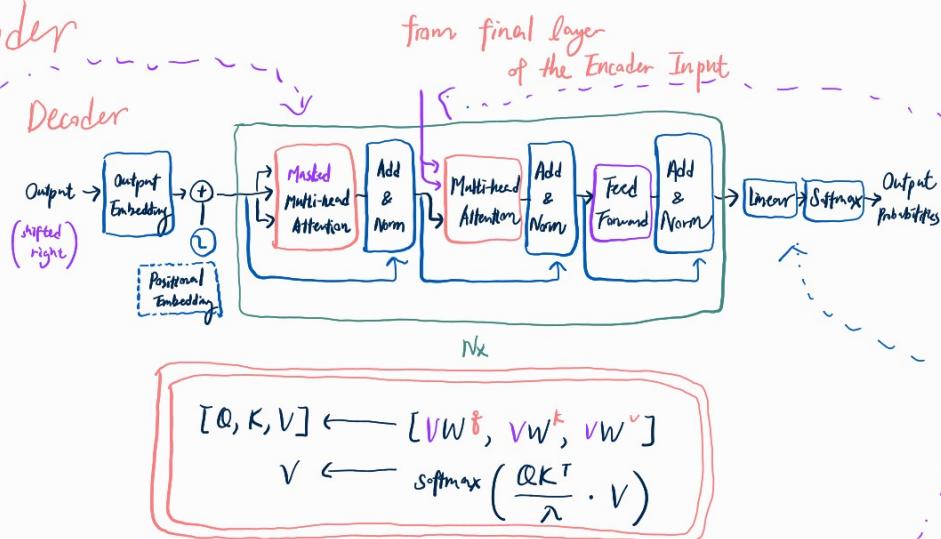
Each layer: (2016)
 Add Residual Connection &
 Layer-wise Normalization

$V \leftarrow VW^T$ Self-Attention
 $V \leftarrow \text{softmax}\left(\frac{VV^T}{\lambda}\right) \cdot V$

- ① 1×1 convolution with d filters
 $W^V = [\vec{w}_1, \dots, \vec{w}_d]$ s.t. $u_{ik} \leftarrow \langle \vec{v}_i, \vec{w}_k \rangle$
- ② Global weighted average pooling
 $\vec{v}_j \leftarrow \sum_i p_{ij} \cdot \vec{u}_i$ where $p_{ij} \propto \exp\left(\frac{\langle \vec{u}_i, \vec{v}_j \rangle}{\lambda}\right)$
- ③ 1×1 convolution with $4d$ filters
 $W = [\vec{w}_1, \dots, \vec{w}_{4d}]$ s.t. $u_{ik} \leftarrow \langle \vec{v}_i, \vec{w}_k \rangle$
- ④ Activation with GELU
- ⑤ 1×1 convolution with d filters again

$\hat{W}_1 \in \mathbb{R}^{d \times 4d}$
 $\hat{W}_2 \in \mathbb{R}^{4d \times d}$ } weights
 $\sigma(t) := \text{GELU}$

Decoder



Masked Self-Attention

each output only depends on input and previous outputs
 \rightarrow set $\langle q_i, k_j \rangle = -\infty$ if $i < j$
 \rightarrow or shift output to the right by 1 position,
 then enforce for $i < j$

Context Attention

Q from the Decoder
 (K, V) from the Encoder Input

\rightarrow each output attends to every position in the input

Last Linear Transformation and Softmax

$$\hat{Y} = \text{softmax}(\hat{V}(\hat{W}^e)^T), \quad \hat{W}^e = \text{transpose of the input encoding}$$

Training Loss

$$\min_w \hat{E}[-\langle Y, \log \hat{Y} \rangle]$$

minimize the multi-class cross-entropy loss

$Y = [\hat{y}_1, \dots, \hat{y}_e]$ output sequence, $\hat{Y} = [\hat{y}_1, \dots, \hat{y}_e]$ predicted probabilities
 \hat{y}_e depends on
 ① the input sequence $X = [\vec{x}_1, \dots, \vec{x}_m]$
 ② the previous output tokens $\hat{y}_1, \dots, \hat{y}_{e-1}$

To accelerate training,

shift the ground-truth (instead of generated) output sequence 1 position to the right,
 s.t. each output symbol, when generated sequentially, only depends on symbols before it.

Comparison between Transformers, RNN, CNN

Layer Type	Complexity (per-layer)	Segmental Operation	Max. Path Length
Self-Attention	$O(m^2d)$	$O(1)$	$O(1)$
Self-Attention (Restricted)	$O(rmd)$	$O(1)$	$O(\frac{m}{r})$
Recurrent NN	$O(md^2)$	$O(m)$	$O(m)$
Convolution NN	$O(kmd^2)$	$O(1)$	$O(\log_k m)$

m : length of the input sequence
 d : internal representation dimension
 max path length = max. no. of sequential operations
 for any output token to attend to any input position

Self-Attention:

dot-product QQ^T costs $O(m^2d)$ since $Q \in \mathbb{R}^{md}$

✓ parallelizable

Self-Attention (Restricted) when quadratic time/space is a concern

✓ good with visualization

attention only to r neighbors instead of m neighbors $\rightarrow O(rmd)$

(attention distribution over layers/positions)

at the cost = ↑↑ max. path length to $O\left(\frac{m}{r}\right)$

Suitable for modelling long-range dependencies

Recurrent NN:

$$\vec{h} \leftarrow W_2 \delta(W_1(h, x)) \text{ costs } O(d^2), \text{ total } O(md^2)$$

Convolution NN:

k : filter size, each convolution costs $O(kd)$

single output filter = need to repeat m times

total d output filters $\Rightarrow O(kmd^2)$

Separable Convolution (2017) = reduce to $O(kmd + md^2)$

✓ parallelizable

For max path length,

dilated convolution (2016): $O(\log_k m)$

usual convolution with stride k : $O\left(\frac{m}{k}\right)$

Transformer Practicalities

Optimizer: Adam (2015) $\beta_1 = 0.9$ $\beta_2 = 0.98$ $\epsilon = 10^{-9}$

Learning Rate: $\gamma_{\text{iter}} = \frac{1}{Jd} \cdot \min \left\{ \frac{1}{\sqrt{\text{iter}}}, \text{iter} \cdot \frac{1}{T \cdot C} \right\}$

$T = 4000$ controls the warm-up stage

where the learning rate increases linearly

After that, decreases inversely proportionally to the square root of iteration no.

Regularization:

(a) Dropout (rate 0.1)

after the positional encoding

& after each attention layer

(b) Residual connection & Layer normalization

after each attention layer

& each Feed-Forward layer

(c) Label Smoothing (2016)

$$\vec{y} \leftarrow (1 - \alpha) \vec{y} + \alpha \frac{1}{C}$$

$\underbrace{\vec{y}}$ original label vector
 $\underbrace{\vec{y}}$ typical one-hot
 α : controls the amount of smoothing
 C : number of classes

Beam Search:

Example: Machine Translation

→ Augment each candidate translation Y_k with a next word $\vec{y} =$

$$\text{score}_k \leftarrow \text{score}_k - \log \hat{P}(\vec{y} | X, Y_k)$$

→ Prune the next word, by considering only those whose score contribution lies close to the best \approx

- Keep only the best b augmented translations
- To reduce bias towards shorter translations, if some candidate translation ends, compute a normalized score $\frac{\text{Score}}{\text{Length}}$

Examples: Image Transformer (2018)
Sparse Transformer (2019)

ELMo (2018)

Embedding from Language Models

Motivation = Conventional: each word has a fixed representation,

ELMo: contextualized

representation for each word depends on the entire sentence (context)

ELMo trains a bidirectional LM

$$\min_{\theta, \bar{\theta}} \hat{E} - \log \vec{p}(X|\theta) - \log \bar{p}(X|\bar{\theta}) \quad \text{where}$$

two probabilities modeled by two LSTMs

$$\begin{cases} \vec{p}(X|\theta) = \prod_{j=1}^m p(x_j | x_1, \dots, x_{j-1}; \theta) & \text{look behind} \\ \bar{p}(X|\bar{\theta}) = \prod_{j=1}^m p(x_j | x_{j+1}, \dots, x_m; \bar{\theta}) & \text{look forward} \end{cases}$$

with shared embedding & softmax layers; but different hidden parameters.

ELMo representation of any token \vec{x} :

$$\text{ELMo}(\vec{x}; A) = A(h_a, \{\vec{h}_e\}_{e=1}^L, \{\bar{h}_e\}_{e=1}^L)$$

Aggregation function between the two 2-layer LSTMs

Total 3 layers of representations

When $b=2$, lower layer: syntactic info
higher layer: semantic info

① context-insensitive through character-level CNN
② forward & backward
③ 2-layer LSTM

Two-Stage (TS) = fixed the parameters in ELMo, use as (additional) feature extractor, on top of which tune a task-specific architecture.
Soft-max layer = last (appears to improve performance)

BERT (2020)

Bidirectional Encoding Representations from Transformers

Follow-up on the pre-training path in GPT (Generative Pre-trained Transformers) with some twists:

Input: Concatenation of two sequences:

[CLS] sequence A [SEP] sequence B [SEP]

Final Representation:

Sentence-level abstraction for downstream sentence

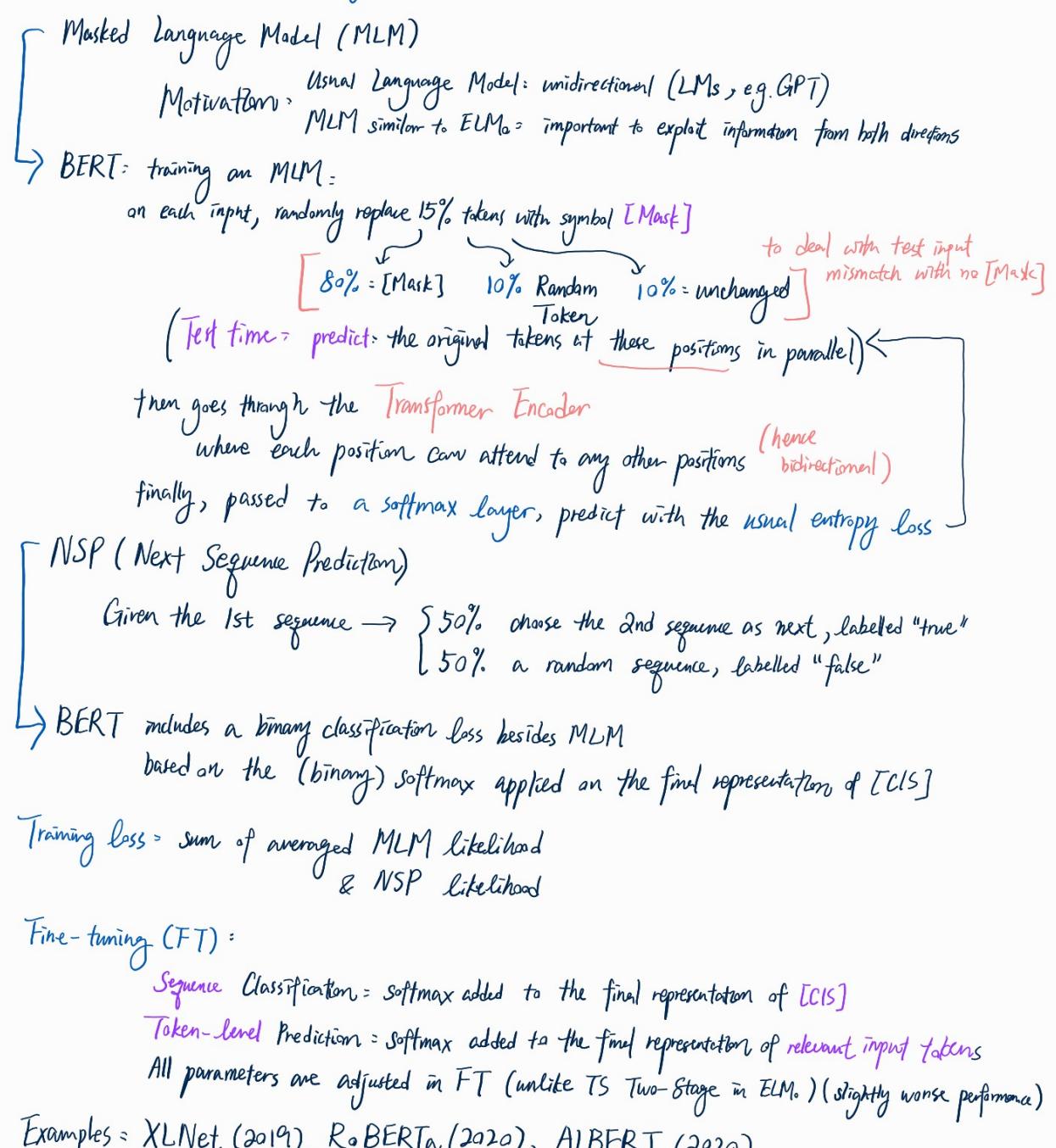
classification tasks: two sequence input format ✓ question answering

Sequence Positional Embedding:

(on top of position embedding)

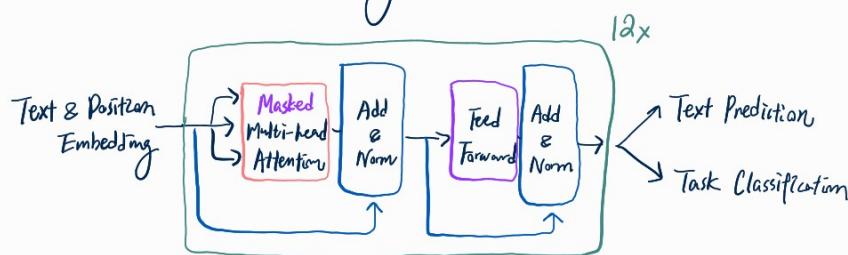
Tokens in 1st sequence share an embedding vector;

— 2nd sequence share another.



GPT (2018)

Generative Pre-Training



→ Unsupervised pre-training through a LM:

$$\min_{\theta} -\hat{E} \log p(X|\theta) \quad \text{where } p(X|\theta) = \prod_{j=1}^m p(\vec{x}_j | \vec{x}_1, \dots, \vec{x}_{j-1}, \theta)$$

probabilities modelled by given context of previous tokens, predict current token

Multi-Layer Transformer Decoder

$$H^{(0)} = XW^e + WP$$

$(\vec{x}_1, \dots, \vec{x}_m)$ input sequence of m tokens (very long)

token embedding matrix

position embedding matrix

$$H^{(l)} = \text{transformer-decoder block}(H^{(l-1)})$$

$$\rightarrow p(x_j | x_1, \dots, x_{j-1}; \theta) = \text{softmax}(\vec{h}_j^{(L)} W^T)$$

$x=1, \dots, L$, number of transformer blocks

\rightarrow Supervised Fine-Tuning with task-aware input transformation:

$$\min_{W^T} \min_{\theta} -\hat{\mathbb{E}} \log p(\vec{y} | X, \theta) - \lambda \cdot \hat{\mathbb{E}} \log p(X | \theta)$$

where $p(\vec{y} | X, \theta) = \langle \vec{y}, \text{softmax}(\vec{h}_m^{(L)} W^T y) \rangle$

include the unsupervised pre-training loss \rightarrow help improve generalization & accelerating convergence

\rightarrow For different tasks, GPT only add an extra softmax layer for classification

Unlike ELMo, GPT avoids task-specific architecture \rightarrow only aim to learn universal representation for all tasks

GPT practicalities

BPE: Byte pair encoding (2016): a practical middle ground between character-level LMs (2016) vs (less competitive) word-level LMs (2019)

Start with UTF-8 bytes (256 base vocabs)

repeatedly merge pairs with the highest frequency in the corpus (e.g. dog and dog!)

✓ compute probabilities over unseen words and sentences

GELU: Gaussian Error Linear Unit (2016)

$$\delta(x) = x \Phi(x) = \mathbb{E}(x \cdot m | x) \quad \text{where } m \sim \text{Bernoulli}(\Phi(x))$$

On the input x , with probability $\Phi(x)$ we drop it.

Unlike ReLU that drops all negative input, GELU drops more probably as the latter decreases

Fine-Tuning: smaller batch size 32 (learning rate)
 converges after 3 epochs warm-up $T=2000$ during pre-training
 $\lambda = \frac{1}{2}$ 0.2% during fine-tuning

GPT-2 (2019)

GPT: unsupervised pre-training + supervised fine-tuning

GPT-2: completely unsupervised zero-shot setting
 + (sharp) increase in model capacity
 + larger training set

GPT-3 (2020)

Crystallization of different evaluation schemes:

Fine-Tuning (FT) = explored in GPT-1

GPT-3 focus involves task-dedicated datasets and gradient updates on both the LM and the task-dependent classifier never for GPT-3

Few-Shot (FS) = a natural language description of the task,
 "Translate English to French"
 a few example demonstrations,

English-French sentence pairs full context (that will be completed by GPT-3)

- the input English sentence
- One-Shot (1S) : FS with no. of examples = 1
 - Zero-Shot (0S) : only provide task description

Examples = Pixel GPT (2020)

Vision Transformers (2021)

LoRA (2022)

Instruct GPT (2022) $\xrightarrow{\text{Alignment}}$ GPT-4 (OpenAI, 2023)

① Collect demo data, train a Supervised Policy

- Sample a prompt
- A labeler demo the desired output behavior
- Data fine-tune GPT-3 with supervised learning

② Collect comparison data, train a reward model

- Sample a prompt and several model outputs
- A labeler ranks the output from best to worse
- Data train the reward model

③ Optimize on policy against the reward model with reinforcement learning

- Sample a new prompt

- Policy generates output
- Reward model calculates a reward for the output
- Reward updates the policy

Lec 12 Graph Neural Networks (later)

Lec 13 Decision Trees

can represent any boolean functions

Classification Tree : leaf nodes represent probability of label

Regression Tree : leaf nodes represent average of responses

Learning : what to put at the leaves?

which variable to split at each stage?

what threshold to use?

when to stop?

→ regularization e.g. early stopping,

→ pruning

Splitting by Minimizing

$$(\hat{j}^*, t^*) = \underset{j=1, \dots, d}{\operatorname{argmin}} \min_{t \in T_j} \left(l(\{\vec{x}_i, y_i\} : x_{ij} \leq t) + l(\{\vec{x}_i, y_i\} : x_{ij} > t) \right)$$

Partition the training data
into two disjoint parts

Greedyly choose the j -th feature to split
 $x_{ij} \leq t$ vs $x_{ij} > t$
 Greedily choose a threshold $t \in T_j$ to split

Stopping Criterion

max. depth exceeded all children nodes are (sufficiently) homogeneous

max. runtime exceeded all children nodes have too few training examples

Reduction in cost stagnates:

$$\Delta := l(D) - \left(\underbrace{\frac{|D_L|}{|D|} \cdot l(D_L)}_{\text{Cross Validation}} + \underbrace{\frac{|D_R|}{|D|} \cdot l(D_R)}_{\text{Right}} \right)$$

Regression Cost

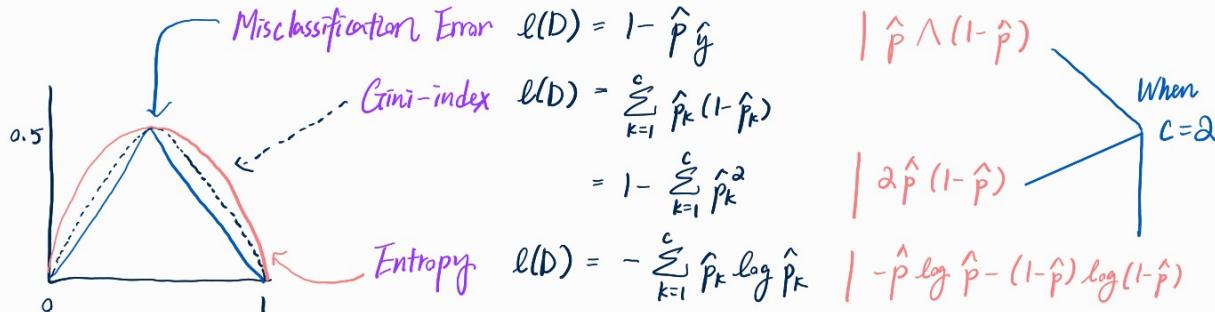
$$l(D) := \left[\min_y \sum_{y_i \in D} (y_i - y)^2 \right]$$

Can use any reasonable loss other than the square loss

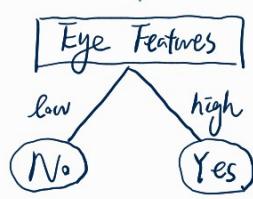
$$= \sum_{y_i \in D} (y_i - \bar{y})^2 \quad \text{where } \bar{y} = \frac{1}{|D|} \sum_{y_i \in D} y_i$$

Classification Cost

$$\hat{p}_k = \frac{1}{|D|} \sum_{y_i \in D} [\mathbb{I}[y_i \in k]] \quad \hat{y} = \arg \max_{k=1, \dots, c} \hat{p}_k$$



Decision Stump



- A binary tree with stump 1
- Performs classification based on 1 feature
- Easy to train, interpretable but underfits (tackle in next lecture)

Lec 14 Boosting vs Bagging (Ensembling)

Motivation: combine a bunch of classifiers to get better performance,
 e.g. when compared against the best classifier in the gang

Condorcet's Jury Theorem (1785)

If we can combine a large number of classifiers with conditionally independent accuracy, and each slightly better than random guessing, we can approach perfect accuracy =

A collection of binary classifiers

$$\{h_t: X \rightarrow \{0, 1\} \text{ binary label}, t=1, 2, \dots, 2T+1\}$$

Conditioned on some auxiliary information F .

Assume (independent) accuracy on test sample (X, Y)

$$p := \Pr(h_t(X) = Y | F) > \frac{1}{2}$$

Assume "Meta" classifier h :

$$h(x) = \begin{cases} 1 & \text{if } \#\{t : h_t(x) = 1\} \geq T+1 \\ 0 & \text{if } \#\{t : h_t(x) = 0\} \geq T+1 \end{cases}$$

Then accuracy:

$$\begin{aligned} & \Pr(h(x) = Y | F) \\ &= \Pr\left(\sum_{t=1}^{2T+1} [h_t(x) = Y] \geq T+1 \mid F\right) \quad \begin{array}{l} \text{increases as } p \\ \propto T \text{ (provided that } p > \frac{1}{2}\text{)} \end{array} \\ &= \sum_{k=T+1}^{2T+1} \binom{2T+1}{k} p^k (1-p)^{2T+1-k} \quad \begin{array}{l} \text{approaches 1 as } T \rightarrow \infty \\ = \frac{(2T+1)!}{(T!)^2} \int_0^p u^T (1-u)^T du \end{array} \end{aligned}$$

Bagging (Bootstrap Aggregating) (1996)

Motivation: approximate independent classifiers by training on independent datasets? luxury
Instead: bootstrapping as in cross-validation

Bagging = perturbs the training set by taking a random subset.

Algorithm: Bagging Predictors

Input: training set D , num of repetitions T

Output: meta-predictor h

$\{D_t\}$ not really independent,

but aggregating predictors usually (not always) improves performance

Reduces Variance

For $t=1, 2, \dots, T$ do

- | sample a new training set $D_t \subseteq D$ // in parallel
- | train predictor h_t on D_t // with or without replacement
- | $h \leftarrow \text{aggregate}(h_1, \dots, h_T)$ // classification = majority vote
regression = average

Randomizing Output (2000)

perturbs the training set by adding noise

Regression Task:

| adding small Gaussian noise to each response y_i

Classification Task:

| randomly flip a small portion of labels y_i

Prevent Overfitting

Random Forest (2001)

Motivation = Decision Trees + Bagging

Adding a layer of randomization:

During training, at each internal node,
select a random subset of features
and perform the splitting

Then bagging + aggregate the "randomized" decision trees

Hedging (1997)

Motivation = Horse-Racing Game

Algorithm: Hedging

Input: initial weight vector $\vec{w}_0 \in \mathbb{R}_+^n$
discount factor $\beta \in [0, 1]$

Output: last weight vector \vec{w}_{T+1}

For $t = 1, 2, \dots, T$ do

| learner chooses probability vector $\hat{p}_t = \frac{\vec{w}_t}{\vec{w}_t^\top \vec{w}_t}$
environment chooses loss vector $l_t \in [0, 1]^n$

| learner suffers (expected) loss $\langle \hat{p}_t, l_t \rangle$

| learner updates weights $\vec{w}_{t+1} = \vec{w}_t \odot \beta^{l_t}$

optional: scaling $\vec{w}_{t+1} \leftarrow c_{t+1} \vec{w}_{t+1}$ ← prevent weight from underflowing

n horses in a race, repeat for T rounds

Each round bet a fixed amount of money,
with

p_i = proportion on horse i at the t -th round

At the end of round t : arbitrary, no iid assumption,
a loss $l_t \in [0, 1]$ that we suffer on horse i

(can assume bounded)

(can be adversarial)

✓ Hedging Guarantee

As $T \rightarrow \infty$, Hedging can compete against the best horse in hindsight

Average loss no larger than that
plus a constant term $\propto \sqrt{\frac{2 \ln n}{T}}$

AdaBoost (1997)

Motivation: Adaptive discount factor $\beta_t \leq 1$

Algorithm: Adaptive Boosting

Input: initial weight vector $\vec{w}_0 \in \mathbb{R}_{++}^n$

$$\text{training set } D_n = \left\{ (\vec{x}_i, y_i) \right\}_{i=1}^n \subseteq \mathbb{R}^d \times \{0, 1\}$$

Output: "meta"-classifier $\bar{h}: \mathbb{R}^d \rightarrow [0, 1]$

$$\vec{x} \mapsto \left[\sum_{t=1}^T \left(\ln \frac{1}{\beta_t} \right) (h_t(\vec{x}) - \frac{1}{2}) \geq 0 \right] \quad \begin{array}{l} \text{* Weighted aggregation first, then threshold} \\ \text{(better in practice)} \end{array}$$

For $t=1, 2, \dots, T$ do

$$\vec{p}_t = \frac{\vec{w}_t}{\vec{1}^\top \vec{w}_t} \quad \begin{array}{l} \text{learner chooses probability vector } \vec{p}_t \\ \text{// normalization} \end{array}$$

$$h_t \leftarrow \text{WeakLearn}(D_n, \vec{p}_t) \quad \begin{array}{l} \text{// } t\text{-th weak classifier } h_t: \mathbb{R}^d \rightarrow [0, 1] \end{array}$$

$$l_t, l_{it} = 1 - |h_t(\vec{x}_i) - y_i| \quad \begin{array}{l} \text{environment chooses loss vector } l_t \\ \text{loss is higher if prediction is more accurate!} \end{array}$$

$$E_t = 1 - \langle \vec{p}_t, l_t \rangle$$

$$= \sum_{i=1}^n p_{it} |h_t(\vec{x}_i) - y_i| \quad \begin{array}{l} \text{learner suffers (expected) loss } \langle \vec{p}_t, l_t \rangle \\ \text{// weighted (expected) error } E_t \in [0, 1] \end{array}$$

$$\beta_t = \frac{E_t}{(1-E_t)} \quad \begin{array}{l} \text{// Adaptive discounting parameter } \beta_t \leq 1 \Leftrightarrow E_t \leq \frac{1}{2} \end{array}$$

$$\vec{w}_{t+1} = \vec{w}_t \odot \beta_t^{l_t} \quad \begin{array}{l} \text{learner updates weights } \vec{w}_{t+1} \\ \text{// element-wise product } \odot \text{ and power} \end{array}$$

* if h_t predicts correctly on training sample \vec{x}_i ,

then suffer a larger loss l_t : st. next iteration assign less weight to \vec{x}_i

i.e. each classifier focuses on hard examples that are misclassified by previous classifier.

✓ AdaBoost Guarantee

AdaBoost → Real-time face detection (2004)

Bagging vs Boosting

Both train an ensemble of weak classifiers to produce a "meta"-classifier

Bagging

✓ parallelization

→ Resamples training examples

→ Averaging independent classifiers
to reduce variance

Boosting

✗ Strictly sequential

→ Longer Training Time

→ Reweights training examples

→ Less Interpretable

→ Averaging dependent classifiers

which might be analyzed through dynamic systems and ergodic theory

Extensions = Logit Boost, Grad Boost, L2Boost, XGBoost

Combining Bagging and Boosting =

e.g. each classifier in Boosting is obtained via Bagging (2000)