Author:
kiking

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 equity?

Lectures:

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

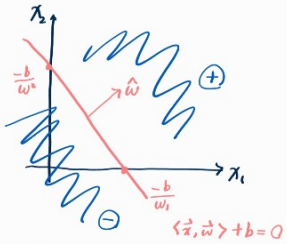
Lec 01 Perceptron

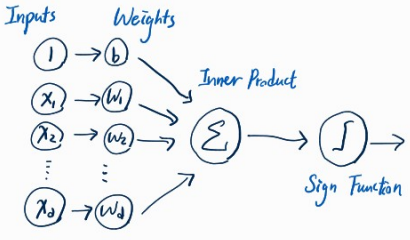
Batch Learning vs Online Learning

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

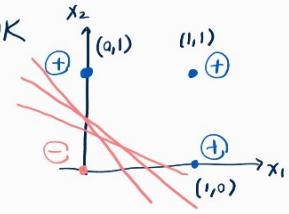
Linear Threshold Function (LFT)

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





OR Dataset → OK



Perceptron Algorithm (1958 by Rosenblatt)

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

Initialization: $\vec{w} \in \mathbb{R}^d, b \in \mathbb{R}$ } typically $\vec{w} = 0, b = 0, \delta = 0$

Threshold: $\delta \geq 0$

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

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

  let $k \in \{1, \dots, n\}$ be an index

  If $y_k \cdot (\langle \vec{x}_k, \vec{w} \rangle + b) \leq \delta$ then

    $\vec{w} \leftarrow \vec{w} + y_k \cdot \vec{x}_k$

    $b \leftarrow b + y_k$

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

Solves Optimization Problem: Find $\vec{w} \in \mathbb{R}^d, b \in \mathbb{R}$ s.t.

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

Key Insight: When a mistake happen,

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

Author:
kiking

Notation Simplification: $y[\langle \vec{x}, \vec{w} \rangle + b] = \langle y(\overbrace{\vec{x}}^{\text{padding}}), \overbrace{(\vec{w}}^{\vec{a}}, \overbrace{b)}^{\hat{w}} \rangle$

Problem becomes: Find $\hat{w} \in \mathbb{R}^p$ s.t. $A^T \hat{w} \underset{\text{element-wise}}{>} \vec{0}$, 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 $\forall i, \langle y_i \vec{x_i}, \hat{w} \rangle > > \gamma$.

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)

Perceptron vs Support Vector Machine vs Soft-margin Support Vector Machine
* Unique Solution
* Beyond Seperability

Lec 02 Linear Regression

Regression: Given dataset $\{(x_i, y_i)\}$, find $f: X \to Y$ s.t. $f(x_i) \approx y_i$

Statistical Learning

i.i.d: Independently and Identically drawn samples from a distribution $\mathbb{P}$
$(X_i, Y_i) \sim \mathbb{P}$ and $(X, Y) \sim \mathbb{P}$

Assumption: Law of Large numbers in Expectation

Regression Function: $m(x) = E[Y | X = \vec{x}]$ approx. well

Least Squares Regression: $\min_{X \to Y} E \| f(X) - Y \|_2^2$

→ Sampling (Training): $\frac{1}{n} \sum_{i=1}^{n} \| f(X_i) - Y_i \|_2^2$

$= E \| f(X) - m(X) \|_2^2 - E \| m(X) - Y \|_2^2$

Approx. Error (bias²) — Aim: $f \approx m$, minimize bias

Estimation Error (noise variance) — does not depend on f, inherit measure of the difficulty of the problem

Linear Least Square Regression: assume $m(x)$ must be linear/affine function

Padding + Matrix Form: $\min_{\hat{w} \in \mathbb{R}^{t \times (d+1)}} \frac{1}{n} \| \underbrace{\hat{w} \hat{X}}_{f(x)} - \hat{Y} \|_F^2$ $\quad \hat{x} = \begin{pmatrix} \vec{x} \\ 1 \end{pmatrix}$, $\hat{w} = [w, b]$

$\| A \|_F = \sqrt{\sum_{ij} a_{ij}^2}$ Frobenius Norm

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

Apply Iterative Constraint Algorithms !!!!

$\hat{X}$ can be ill-conditioning e.g. $\hat{X} = \begin{bmatrix} 0 & \varepsilon \\ 1 & 1 \end{bmatrix} \to \hat{w} = y \hat{X}^{-1} = [-\frac{2}{\varepsilon}, 1]$ close to rank deficient

Slight Perturbation leads to Chaotic behaviours

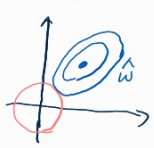
Ridge Regression: $\min_{\hat{w}} \frac{1}{n} \| \hat{w} \hat{X} - \hat{Y} \|_F^2 + \boxed{\lambda \| \hat{w} \|_F^2}$ Regularization ⇔ Constraints $\min_{\|\vec{w}\|_F \le \gamma} \frac{1}{n} \| \hat{w} \hat{X} - \hat{Y} \|_F^2$

Data Augmentation = regularization: $= \frac{1}{n} \| \hat{w} [\hat{X} \ \sqrt{n\lambda} I] - [\hat{Y} \ \hat{0}] \|_F^2$

augments p data points $x_j = \sqrt{n\lambda} e_j$

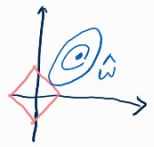
restricts output to be $\frac{1}{\lambda}$ proportion to input

shrinks $\hat{w} \to \hat{0}$ when $\lambda \to \infty$



More computation

Lasso Regression (1996): $\min_{\hat{w}} \frac{1}{n} \| \hat{w} \hat{X} - \hat{Y} \|_F^2 + \boxed{\lambda \| \hat{w} \|_1}$ Regularization ⇔ Constraints $\min_{\|\vec{w}\|_1 \le \gamma} \frac{1}{n} \| \hat{w} \hat{X} - \hat{Y} \|_F^2$



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

Assumption: Tasks are independent and can be solved separately

⇒ Cross Validation: k-fold validation

$\text{Perf}(\lambda) = \text{Perf}_1 + \text{Perf}_2 + \dots + \text{Perf}_k$

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

Lec 03 Logistic Regression

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

Author:
kiking

into probability estimates.

Confidence = $|\vec{w}^T\vec{x}+b|$? hard to interpret

Parametrization of Binary Classification Problem

Posterior Probability: Let $p(\vec{x};\vec{w})$ approx. $Pr(Y=1 \mid X=\vec{x})$ where labels $\vec{y} \in \{\pm 1\}$

where $p(\vec{x};\vec{w}) \in [0,1]$

- Bernoulli Model: $Pr(Y=y \mid X=\vec{x}) = p(\vec{x};\vec{w})^{\frac{(1+y)}{2}} (1-p(\vec{x};\vec{w}))^{\frac{(1-y)}{2}}$
- Conditional Likelihood = $Pr(Y=y_1, \dots, Y_n=y_n \mid X_1=\vec{x}_1, \dots, X_n=\vec{x}_n)$
- Maximize this Conditional Likelihood w.r.t. $\vec{w}$! $= \prod_{i=1}^{n} p(\vec{x}_i;\vec{w})^{\frac{(1+y_i)}{2}} \left(1-p(\vec{x}_i;\vec{w})\right)^{\frac{(1-y_i)}{2}}$ → 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 $[0,1]$; $\langle \vec{x},\vec{w}\rangle + b \to \mathbb{R}$.

Logit Transformation

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

equivalently $p(\vec{x};\vec{w}) = \frac{\exp(\langle \vec{x},\vec{w}\rangle + b)}{1+\exp(\langle \vec{x},\vec{w}\rangle + b)}$ Big Assumption!

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

predict confidence of $\vec{x}$ given trained $\vec{w}, b$

Logistic Loss (after plugging into the Likelihood equation)

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

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

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
- Softmax Regression

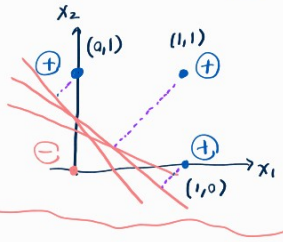
$\hat{p}_i = f(\hat{W}\vec{x}_i) = \text{softmax}(\hat{W}\vec{x})$, $\text{softmax}: \mathbb{R}^C \to \Delta_{C-1}$,

$y \mapsto \frac{\exp(y)}{\vec{1}^T \exp(y)}$

Lec 04 Support Vector Machines

Motivation: Maximizing Margin, ie.

Maximizing the minimum distance to every point of a hyperplane



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

Quadratic Programming (Perceptron = Linear Programming)

Unique Solution (Perceptron: infinite solutions)

$\min_{\vec{w},b} 0$ s.t. $y_i\hat{y}_i > 0 \ \forall i$ Primal form

Author:
kiking

Support Vectors: points lie on the hyperplane (decisive)

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

$$(1-a)\vec{x} + a\vec{z} \in C$$

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

Example: ϕ, $\mathbb{R}^d$ (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.

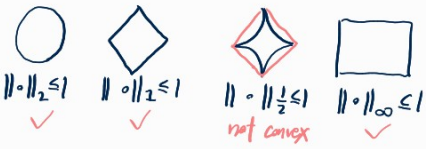
$$\text{conv}(A) := \bigcap_{\text{convex } C \supseteq A} C$$

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 $\vec{x}_1, \ldots, \vec{x}_n$ as

$$\vec{x} = \sum_{i=1}^{n} a_i \vec{x}_i \quad \text{where } \vec{a} \geq 0 \ \ \vec{1}^T \vec{a} = 1$$

i.e. $\vec{a} \in \Delta_{n-1}$

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

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

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

Dual View of SVM (1965)

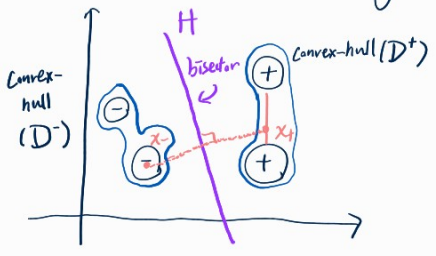
A dataset D is (strictly) linearly separable

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

$$\min_{\vec{w}} \frac{1}{2}\|\vec{w}\|_2^2 + \boxed{C \cdot \sum_{i=1}^{n} (1 - y_i \hat{y}_i)^+} \quad \text{s.t. } \hat{y}_i = \langle \vec{x}_i, \vec{w} \rangle + b \quad \forall i$$

Author:
kiking

w, b

margin maximization

$\sum_{i=1}^{n}$ i-th training error, 0 if $y_i \hat{y}_i \geq 1$

hinge loss $(1-y\hat{y})^+$

Soft Penalty: the more deviate, the heavier the penalty

Update Rule:

Gradient Descent: $\vec{w} \leftarrow \vec{w} - \eta \left[\frac{1}{n} \sum_{i=1}^{n} \ell'(y_i \hat{y}_i)\, y_i \vec{x}_i + \frac{\vec{w}}{\lambda} \right]$ O(nd); ℓ' = derivative of loss function

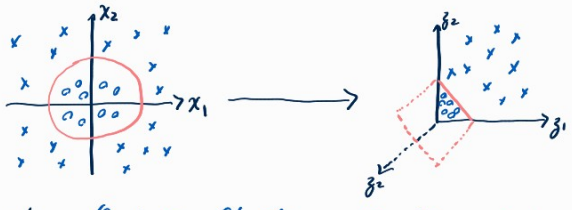
Stochastic Gradient: $\vec{w} \leftarrow \vec{w} - \eta \left[\ell'(y_I \hat{y}_I)\, y_I x_I + \frac{\vec{w}}{\lambda} \right]$ O(d); I = (random sample suffices)

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

- ✓ 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!

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

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

Weights: $Q \in \mathbb{R}^{d \times d}$, $\vec{p} \in \mathbb{R}^d$, $b \in \mathbb{R}$

Linear when $Q = \vec{0}$

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

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

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

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

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

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

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

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

Choose ϕ → determines k

determines some ϕ ← Choose k

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

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

decides a unique RKHS (reproducing kernels hilbert space)

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

Verify a Kernel

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

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

Author:
kiking

Some form of similarity measure

$K_{ij} = K_{ji}$

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

Examples:

Polynomial kernel: $k(\vec{x}, \vec{z}) = (\langle \vec{x}, \vec{z} \rangle + 1)^p$

Gaussian kernel: $k(\vec{x}, \vec{z}) = \exp\left(\frac{-\|\vec{x} - \vec{z}\|_2^2}{\sigma}\right)$ infinite-dimensional RKHS

Laplace kernel: $k(\vec{x}, \vec{z}) = \exp\left(\frac{-\|\vec{x} - \vec{z}\|_2}{\sigma}\right)$

Brownian 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

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

$\langle \vec{x}_i, \vec{x}_j \rangle \to k(\vec{x}_i, \vec{x}_j)$

no need to know ϕ

Testing: $f(\vec{x}) := \langle \phi(\vec{x}), \vec{w} \rangle = \left\langle \phi(\vec{x}), \sum_{i=1}^{n} \alpha_i y_i \phi(\vec{x}_i) \right\rangle = \sum_{i=1}^{n} \alpha_i y_i k(\vec{x}, \vec{x}_i) \in \mathcal{H}_k$

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

New	Old
Training: $O(n^2 d)$	Training: $O(nd)$
Testing: $O(nd)$	Testing: $O(d)$

Trade-Off

- ✓ avoid explicit dependance on feature dimension (e.g. ∞)
- ✗ n times slower in both training + testing
- ✗ need to store training set (in case of SVM)

Kernel Logistic Regression

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

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

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

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

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 $\vec{w}$

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

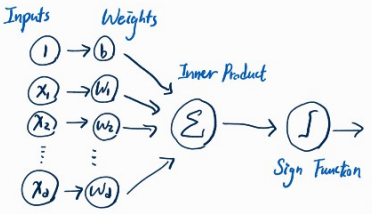
$$\Rightarrow \min_{\vec{w}, \beta} \frac{1}{n} \sum_i \ell\left(\left\langle \bigoplus_p \beta_p \phi_p(\vec{x}_i), \vec{w} \right\rangle\right) + \frac{\lambda}{2} \|\vec{w}\|_2^2$$

Author:
kiking

Representer Theorem $\vec{w} = \sum_j \alpha_j \oplus_p \beta_p \phi_p(\vec{x}_j)$

Lec of Multi-Layer Perceptron

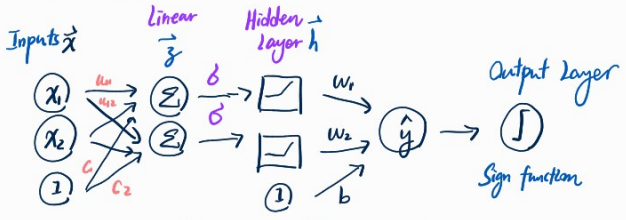
Recall: Perceptron



Algorithm: for $t=1,2,\dots$ do
  Let $k \in \{1,\dots,n\}$ be an index
  If $y_k \cdot (\langle \vec{x}_k, \vec{w} \rangle + b) \le 0$ then
    $\vec{w} \leftarrow \vec{w} + y_k \vec{x}_k$
    $b \leftarrow b + y_k$

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

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



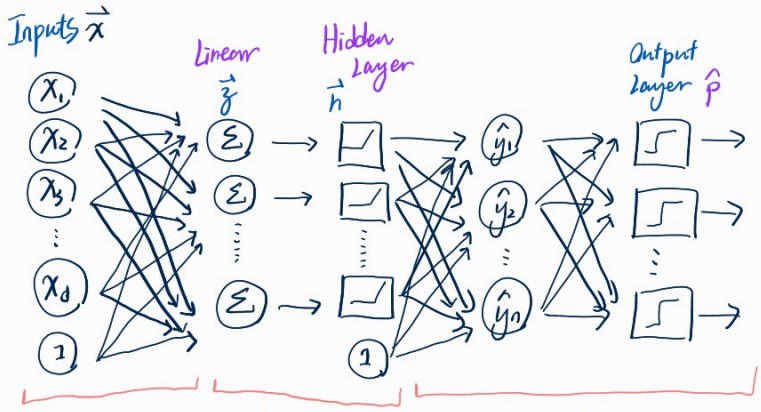
$\vec{z} = U\vec{x} + c$ $\quad \vec{h} = \sigma(\vec{z})$ $\quad \hat{y} = \langle \vec{h}, \vec{w} \rangle + b$

$U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} \in \mathbb{R}^{2\times 2}$, $c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \in \mathbb{R}^2$

element-wise non-linear activation

1st linear transformation — 2nd linear transformation — makes all the difference!

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



$\vec{z} = U\vec{x} + c$ $\quad \vec{h} = \sigma(\vec{z})$ $\quad \hat{y} = Wh + b$ $\quad \hat{p} = \text{softmax}(\hat{y})$

Repeat: hierarchically (Depth) ⇓

learned nonlinear feature map ϕ and the hyperplane (solution)

linear predictor

Algorithm (Test time)

$h_0 \leftarrow \vec{x}$

for $k = 1, \dots, \ell$ do (feature map: layer by layer)
  $z_k \leftarrow W_k h_{k-1} + b$
  $h_k \leftarrow \sigma(z_k)$

$\hat{y} \leftarrow W_{\ell+1} h_\ell + b_{\ell+1}$

$\hat{p} \leftarrow \text{softmax}(\hat{y})$

Activation Functions


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


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


$\text{Relu}(t) = t_+$


$\text{elu}(t) = (t)_+ + (t)_- \cdot (\exp(t) - 1)$

MLP Training

Author:
kiking

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,

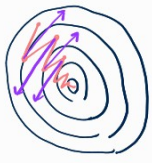
$$w \leftarrow w - \eta \cdot \underbrace{\frac{1}{|B|} \sum_{i \in B}}_{\text{instead of } \frac{1}{n}\sum_{i=1}^{n} \text{ (full pass over the training set)}} \nabla \left[\ell(f(\vec{x}_i; \vec{w}), y_i) \right]$$

Trade-off = variance ↑↑
computation ↓↓

Momentum

Interpretation: gradient averaging

$$\vec{w}_{t+1} = \underbrace{\vec{w}_t - \eta_t \nabla g(\vec{w}_t)}_{\text{gradient step}} + \underbrace{\beta_t (\vec{w}_t - \vec{w}_{t-1})}_{\text{momentum}}$$



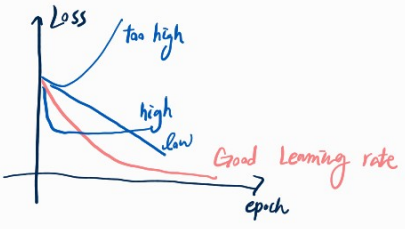
Nesterov Momentum: averaging gradients looking ahead

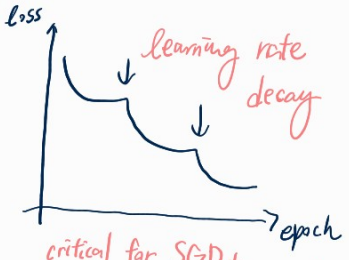
$$\vec{w}_{t+1} = \vec{z}_t - \eta_t \nabla g(\vec{w}_t), \quad \vec{z}_{t+1} = \underbrace{\vec{w}_{t+1} + \beta_t (\vec{w}_{t+1} - \vec{w}_t)}_{\text{looks ahead}}$$

Adjust Learning Rate

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

	s_t	$\vec{v}_t$
Adagrad (Adaptive Gradient)	$s_{t-1} + \nabla g(\vec{w}_t) \odot \nabla g(\vec{w}_t)$	$\nabla g(\vec{w}_t)$
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)	$(1-\lambda_t) s_{t-1} + \lambda_t \nabla g(\vec{w}_t) \odot \nabla g(\vec{w}_t)$	$(1-\gamma_t) \vec{v}_{t-1} + \gamma_t \nabla g(\vec{w}_t)$





$$\eta_t = \eta_0 \exp\left(-C \left\lceil \frac{t}{k} \right\rceil\right)$$

$$\left.\begin{aligned} \eta_t &= \frac{\eta_0}{(1+Ct)} \\ \eta_t &= \frac{\eta_0}{\sqrt{(1+Ct)}} \end{aligned}\right\} \text{sublinear decay}$$

critical for SGD+ Momentum (less for Adam)

Computational Graph

DAG: Direct Acyclic Graph

Forward vs Backward Differentiation

Universal Approximation

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be Riemann Integrable on any bounded interval

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

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

Dropout

Author:
kiking

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

Batch Normalization

Input: $\vec{x}$ over a mini-batch: $\mathcal{B} = \{x_1, \dots, x_m\}$
Parameters to be learnt: γ, β
Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

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

Variance: $\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$

Normalize: $\hat{x}_i \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma_B^2 + \varepsilon}}$

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: conv → relu → conv → relu → pool → FC → relu → FC
(conv → relu → pool: repeat; FC → relu: repeat)

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 (lp-) norm pool
size, stride: as in convolution; no parameter
no padding (typically)

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

Residual Block:

$h^{(I)} \rightarrow$ Conv → Relu → Conv → ReLu $\rightarrow h^{(I+2)}$
(a residual block)

Author:
kiking

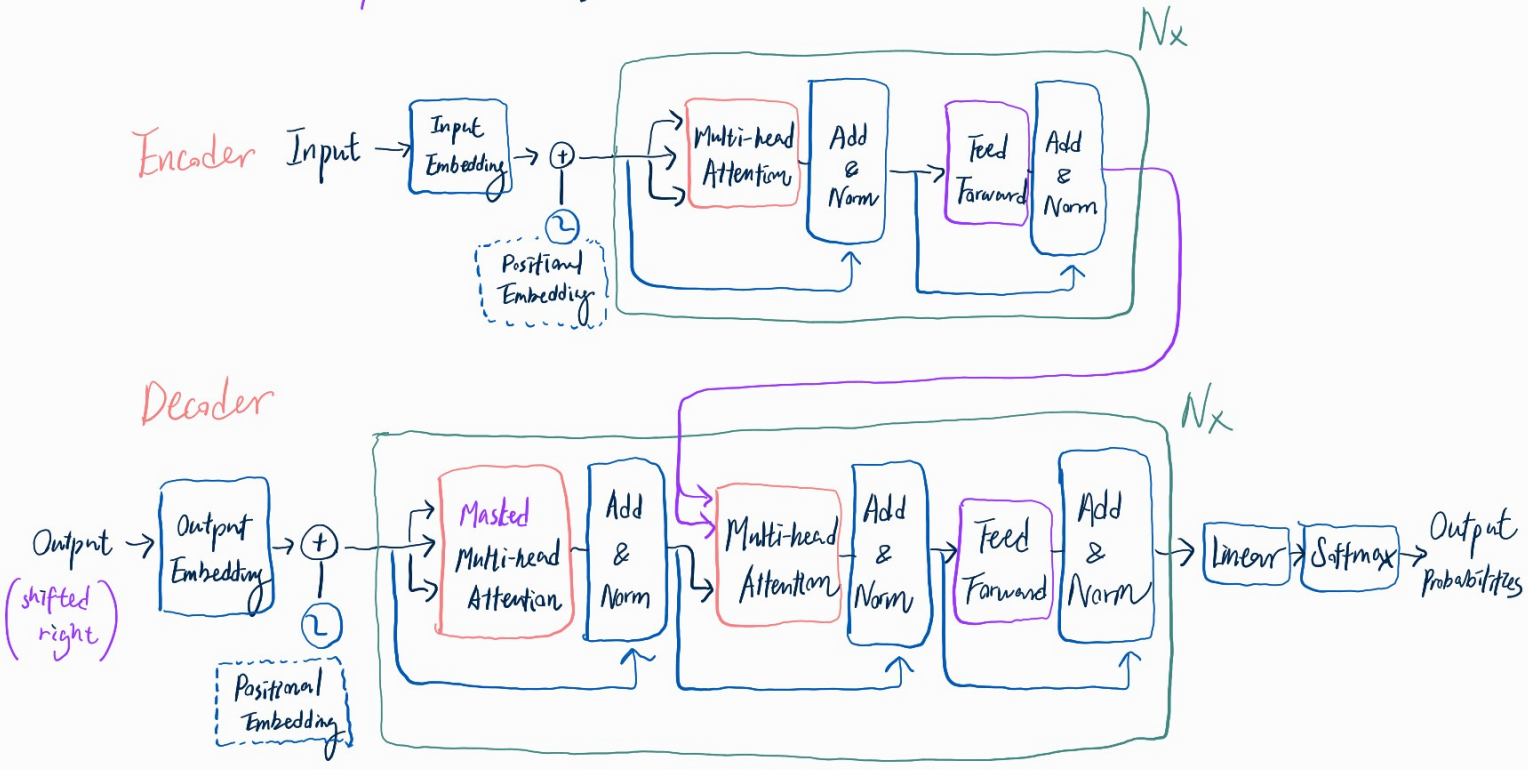
Short-cut connection that allows "skipping" one or more layers
Allow more direct backpropagation of the gradient

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)



Input Sequences $\begin{pmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_m \end{pmatrix}$ Output Sequence $\begin{pmatrix} \vec{y}_1 \\ \vec{y}_2 \\ \vdots \\ \vec{y}_e \end{pmatrix}$

$\vec{x}_m, \vec{y}_e \in \mathbb{R}^p$ one-hot

p: dictionary size

Embedding: XW^e and YW^e, $W^e \in \mathbb{R}^{p \times d}$, $d = 512$

$\begin{pmatrix} \vec{x}_1 \\ \vdots \\ \vec{x}_m \end{pmatrix}_{m \times p} \cdot \underbrace{\begin{pmatrix} \vec{w}_1 & \vec{w}_2 & \cdots & \vec{w}_d \end{pmatrix}}_{p \times d}$

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

(Addictive) Positional Encoding: $W^p \in \mathbb{R}^{len \times d}$

t × d: $2i$, $2i+1$

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

$i = 0, \ldots, \frac{d}{2} - 1$

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

The chosen periodic functions: handle test sequence

Author:
kiking

longer than the training sequences
(Also can direct learn)

Attention (2015)

Motivation: Given a Query vector $\vec{q} \in \mathbb{R}^{d_k}$
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:

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

convex combination of the values in the database

Solution of π_i (coefficient vector):

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

$\text{dist}_K(\vec{q}, \vec{k}_i)$: dissimilarity between the query $\vec{q}$ and the key $\vec{k}$

$\pi_i \log \pi_i$: entropy regularizer

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

Popular choice of dissimilarity function:

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

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

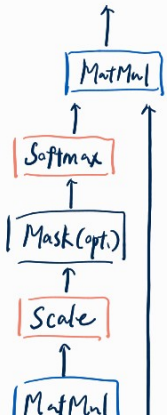
Self-Attention

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

- 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

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



$$[Q, K, V] \leftarrow [QW^Q, KW^K, VW^V]$$
$$V \leftarrow \text{softmax}\left(\frac{QK^T}{\lambda}\right) \cdot V$$

Q: Query, linear transformed by W^Q
K: Key, linear transformed by W^K
V: Value, linear transformed by W^V

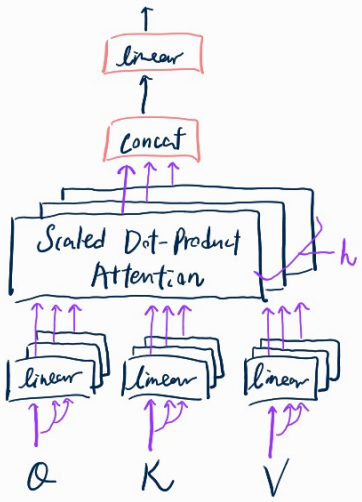
- Dot product between Query Q and Key K: measures similarity
- Output: convex combination of input
- Self-Attention: a special case

Author:
kiking

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

Q K V

✱ Multi-head Attention



$h = 8$ or 12

(keep $d = 512$)

for $i = 1, \dots, h$ do $V_i = att(QW_i^Q, KW_i^K, VW_i^V)$

// linear transformation

$[Q, K, V] \leftarrow [QW_i^Q, KW_i^K, VW_i^V]$

// convex combination

$V_i \leftarrow \text{softmax}\left(\frac{QK^T}{\lambda}\right) \cdot V$

// concatenation & linear terms

$V \leftarrow [V_1, \dots, V_h] W$

$W_i^Q, W_i^K \in \mathbb{R}^{d_k \times d}$

$W_i^V \in \mathbb{R}^{d_v \times d}$

$W \in \mathbb{R}^{(h \cdot d_v) \times d}$

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

$FFN(\hat{V}) = \delta(\hat{V}\hat{W}_1)\hat{W}_2$ ⟶ Two-layer Network

$\hat{V} \in \mathbb{R}^{len \times d}$ → values at some layer, arranged row-wise

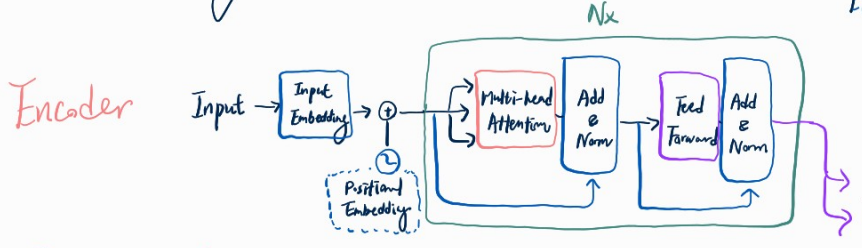
$\hat{W}_1 \in \mathbb{R}^{d \times 4d}$, $\hat{W}_2 \in \mathbb{R}^{4d \times d}$ } weights, shared among different positions, change from layer to layer

$\delta(t) :=$ GELU (Gaussian Error Linear Unit)

$y(t) = t \cdot \Phi(t)$ $\quad$ $y'(t) = t\phi(t) + \Phi(t)$

Encoder: Understanding Attention

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



Self-Attention

$V \leftarrow VW^V$

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

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 $p_{ij} \propto \exp\left(\frac{\langle \vec{u}_j, \vec{u}_i \rangle}{\lambda}\right)$

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

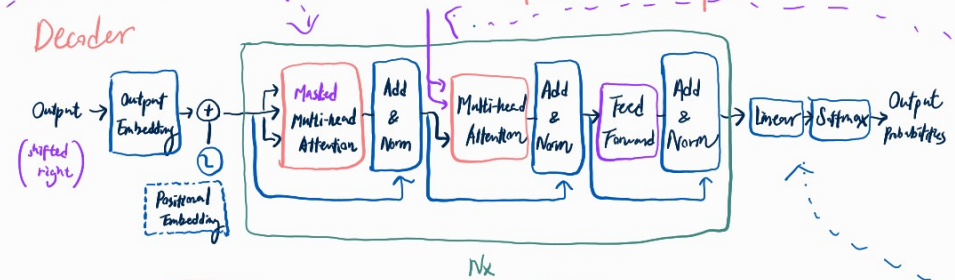
⑤ 1×1 convolution with d filters again

$FFN(\hat{V}) = \delta(\hat{V}\hat{W}_1)\hat{W}_2$

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

Author:
kiking

Decoder



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

$$V \leftarrow \text{softmax}\left(\frac{QK^T}{\lambda} \cdot V\right)$$

Masked Self-Attention

each output only depends on input and previous outputs

→ set $\langle q_i, k_j \rangle = -\infty$ if $i \leq j$

→ 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

→ each output attends to every position in the input

Last Linear Transformation and Softmax

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

Training Loss

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

minimize the multi-class cross-entropy loss

$Y = [\vec{y}_1, \ldots, \vec{y}_e]$ output sequence, one-hot

$\hat{Y} = [\hat{y}_1, \ldots, \hat{y}_e]$ predicted probabilities

$\hat{y}_t$ depends on

① the input sequence $X = [\vec{x}_1, \ldots, \vec{x}_m]$

② the previous output tokens $\vec{y}_1, \ldots, \vec{y}_{t-1}$

To accelerate training,

shift the grand-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	(per-layer) Complexity	Sequential Operation	Max. Path Length
Self-Attention	$O(m^2 d)$ ✓parallelizable	$O(1)$	$O(1)$
Self-Attention (Restricted)	$O(rmd)$	$O(1)$	$O(\frac{m}{r})$
Recurrent NN	$O(md^2)$	$O(m)$	$O(m)$
Convolution NN	$O(kmd^2)$ ✓parallizable	$O(1)$	$O(\log_k m)$

Author:
kiking

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 $O(m^2 d)$ since $Q \in \mathbb{R}^{m \times d}$ ✓ parallizable

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

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

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

Suitable for modelling long-range dependencies

✓ good with visualization (attention distribution over layers/positions)

Recurrent NN:

$\vec{h} \leftarrow W_2 \sigma(W_1(h, x))$ costs $O(d^2)$, 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 $\Rightarrow O(kmd^2)$ ✓ parallizable

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

For max path length,

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

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

Transformer Practicalities

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

Learning Rate: $\eta_{iter} = \frac{1}{\sqrt{d}} \cdot \min\left\{\frac{1}{\sqrt{iter}}, iter \cdot \frac{1}{\tau \cdot \sqrt{\tau}}\right\}$

$\tau = 4000$ controls the warm-up stage where the learning rate increases linearly

After that, decreases inverse 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)

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

$\vec{y}$: original label vector (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 $\vec{y}$:

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

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

Author:
kiking

→ Keep only the best b augmented translations

→ To reduce bias towards shorter translations, if some candidate translation ends, compute a normalized score $\frac{\text{Score}}{\text{Length}^{0.6}}$

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

$$\min_{\theta, \Phi} \; \hat{\mathbb{E}} - \log \overrightarrow{p}(X|\theta) - \log \overleftarrow{p}(X|\Phi) \quad \text{where}$$

two probabilities modeled by two LSTMs

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

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

ELMo representation of any token $\vec{x}$:

$$\text{ELMo}(\vec{x}; A) = A\left(h_0, \{\overrightarrow{h}_\ell\}_{\ell=1}^{L}, \{\overleftarrow{h}_\ell\}_{\ell=1}^{L}\right)$$

Aggregation function; embedding shared between the two L-layer LSTMs; $\overrightarrow{h}, \overleftarrow{h}$ = hidden states of LSTMs

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

Total 3 layers of representation: ① context-insensitive through character-level CNN; ② forward & backward L-layer LSTM; ③

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 Representation 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)

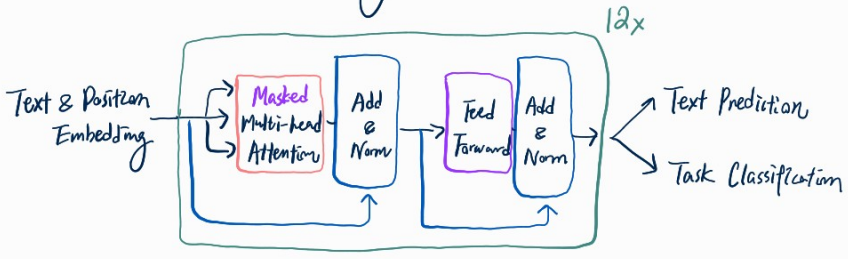
Tokens in 1st sequence share an embedding vector; —|— 2nd sequence share another.

Author:
kiking

- Masked Language Model (MLM)
  - Motivation: Usual Language Model: unidirectional (LMs, e.g. GPT)
    MLM similar to ELMo: important to exploit information from both directions
  - BERT: training an MLM:
    on each input, randomly replace 15% tokens with symbol [Mask]
    [80% : [Mask] 10% Random Token 10% : unchanged] to deal with test input mismatch with no [Mask]
    (Test time: predict the original tokens at these positions in parallel)
    then goes through the Transformer Encoder
    where each position can attend to any other positions (hence bidirectional)
    finally, passed to a softmax layer, predict with the usual entropy loss
- NSP (Next Sequence Prediction)
  - Given the 1st sequence → {50% choose the 2nd sequence as next, labelled "true"; 50% a random sequence, labelled "false"}
  - BERT includes a binary classification loss besides MLM
    based on the (binary) softmax applied on the final representation of [CLS]
- Training loss: sum of averaged MLM likelihood & NSP likelihood
- Fine-tuning (FT):
  - Sequence Classification: softmax added to the final representation of [CLS]
  - Token-level Prediction: softmax added to the final representation of relevant input tokens
  - All parameters are adjusted in FT (unlike TS Two-Stage in ELMo) (slightly worse performance)
- Examples: XLNet (2019), RoBERTa (2020), ALBERT (2020)

GPT (2018)

Generative Pre-Training



→ Unsupervised pre-training through a LM:

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

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

Multi-Layer Transformer Decoder

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

$\begin{pmatrix} \vec{x}_1 \\ \vdots \\ \vec{x}_m \end{pmatrix}$ input sequence of m tokens (var length) — token embedding matrix — position embedding matrix

$$H^{(\ell)} = \text{transformer\_decoder\_block}(H^{(\ell-1)})$$

Author:
kiking

$l = 1, \dots, L$: number of transformer blocks

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

→ Supervised Fine-Tuning with task-aware input transformation:

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

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

where $p(\vec{y} | X, \theta) = \langle \vec{y}, \text{softmax}(h_m^{(l)} W_y) \rangle$

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

Unlike ELMo, GPT avoids task-specific architecture → 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)

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

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 epoches warm-up τ = 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 (never for GPT-3)
involves task-dedicated datasets and [gradient updates on both the LM] and the task-dependent classifier

GPT-3 focus

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)

Author:
kiking

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) ⟶ (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 a 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

Spliting by Minimizing

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

Partition the training data into two disjoint parts

Author:
kiking

Greedily choose the j-th feature to split — $x_{ij} \le t$ vs $x_{ij} > t$

Greedily choose a threshold $t \in T_j$ to split

Stopping Criterion

max. depth exceed — all children nodes are (sufficiently) homogeneous

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

Reduction in cost stagnates:

$$\Delta := \ell(D) - \left(\frac{|D_L|}{|D|} \cdot \ell(D_L) + \frac{|D_R|}{|D|} \cdot \ell(D_R) \right)$$

(Left, Right)

Cross Validation

Regression Cost

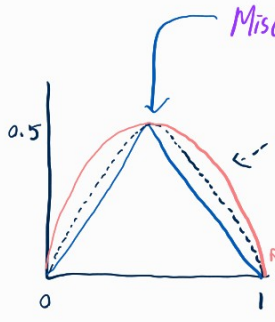
$$\ell(D) := \left[\min_{y} \sum_{y_i \in D} (y_i - y)^2 \right]$$

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

$$\hat{p}_k = \frac{1}{|D|} \sum_{y_i \in D} [\![y_i \in k]\!] \qquad \hat{y} := \arg\max_{k=1,\dots,C} \hat{p}_k$$



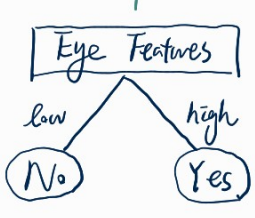
Misclassification Error $\ell(D) = 1 - \hat{p}_{\hat{y}}$ | $\hat{p} \wedge (1-\hat{p})$

Gini-index $\ell(D) = \sum_{k=1}^{C} \hat{p}_k (1-\hat{p}_k) = 1 - \sum_{k=1}^{C} \hat{p}_k^2$ | $2\hat{p}(1-\hat{p})$

When $C=2$

Entropy $\ell(D) = -\sum_{k=1}^{C} \hat{p}_k \log \hat{p}_k$ | $-\hat{p}\log\hat{p} - (1-\hat{p})\log(1-\hat{p})$

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

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

Conditioned on same auxiliary information F.

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

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

Author:
kiking

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:

$$\Pr(h(x)=Y \mid F) = \Pr\left(\sum_{t=1}^{2T+1} [\![h_t(x)=Y]\!] \geq T+1 \;\middle|\; F\right) = \sum_{k=T+1}^{2T+1} \binom{2T+1}{k} p^k (1-p)^{2T+1-k} = \frac{(2T+1)!}{(T!)^2} \int_0^p u^T (1-u)^T \, du$$

increases $\propto p$, $\propto T$ (provided that $p > \frac{1}{2}$), approaches 1 as $T \to \infty$

Bagging (Boostrap Aggregating) (1996)

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

Bagging: perturbs the training set by taking a random subset

Algorithm: Bagging Predictors

Input: training set D, num of reptitions T
Output: meta-predictor h

For $t=1,2,\dots,T$ do // in parallel
  ✱ Sample a new training set $D_t \subseteq D$ // with or without replacement
  train predictor h_t on D_t
$h \leftarrow$ aggregate $(h_1,\dots,h_T)$ // classification: majority vote; regression: average

Reduces Variance

$\#\{D_t\}$ not really independent, but aggregating predictors usually (not always) improves performance

Randomizing Output (2000)

Prevent Overfitting

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

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

n horses in a race, repeat for T rounds
Each round bet a fixed amount of money, with
  p_{it}: proportion on horse i at the t-th round
At the end of round t: arbitrary, no i.i.d assumption
  a loss $l_{it} \in [0,1]$ that we suffer on horse i
  (can assume bounded)
  (can be adversarial)

Algorithm: Hedging

Input: initial weight vector $\vec{w}_1 \in \mathbb{R}^n_{++}$ ← important: strictly pos. init.
  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 $\vec{p}_t = \frac{\vec{w}_t}{1^T \vec{w}_t}$
  environment chooses loss vector $\vec{l}_t \in [0,1]^n$
  learner suffers (expected) loss $\langle \vec{p}_t, \vec{l}_t \rangle$
  learner updates weights $\vec{w}_{t+1} = \vec{w}_t \odot \beta^{\vec{l}_t}$ ← Similar to Perceptron (fix it if it performs badly) but with Multiplicative updates (instead of adding)
  optional: scaling $\vec{w}_{t+1} \leftarrow c_{t+1}\vec{w}_{t+1}$ ← prevent weight from underflowing

✓ Hedging Guarantee

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

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

Author:
kiking

Adaboost (1997)

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

Algorithm : Adaptive Boosting

Input : initial weight vector $\vec{w}_1 \in \mathbb{R}^n_{++}$

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

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

$$\vec{x} \mapsto [\![\sum_{t=1}^{T} (\ln \frac{1}{\beta_t})(h_t(\vec{x}) - \frac{1}{2}) \ge 0]\!]$$

✱ Weighted aggregation first, then threshold (better in practice)

A classifier h_t with lower error ε_t is assigned with a higher weight $\ln \frac{1}{\beta_t} = \ln(\frac{1}{\varepsilon_t} - 1)$ in the final aggregation (make sense)

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

- $\vec{p}_t = \frac{\vec{w}_t}{1^T \vec{w}_t}$ learner chooses probability vector $\vec{p}_t$ // normalization
- $h_t \leftarrow \text{WeakLearn}(D_n, \vec{p}_t)$ // t-th weak classifier $h_t : \mathbb{R}^d \to [0,1]$
- $\forall i$, $\ell_{it} = 1 - |h_t(\vec{x}_i) - y_i|$ environment chooses loss vector ℓ_t // loss is higher if prediction is more accurate!
- $\varepsilon_t = 1 - \langle \vec{p}_t, \ell_t \rangle = \sum_{i=1}^n p_{it} |h_t(\vec{x}_i) - y_i|$ } learner suffers (expected) loss $\langle \vec{p}_t, \ell_t \rangle$ // weighted (expected) error $\varepsilon_t \in [0,1]$
- $\beta_t = \frac{\varepsilon_t}{(1-\varepsilon_t)}$ // Adaptive discounting parameter $\beta_t \le 1 \Leftrightarrow \varepsilon_t \le \frac{1}{2}$
- $\vec{w}_{t+1} = \vec{w}_t \odot \beta_t^{\ell_t}$ learner updates weights $\vec{w}_{t+1}$ // element-wise product $\odot$ and power

✱ if h_t predicts correctly on training sample $\vec{x}_i$,
then suffer a larger loss ℓ_{ti} s.t. next iteration assign less weight to $\vec{x}_i$
i.e. each classier focus 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

✓ parallerization
→ Resamples training examples
→ Averaging independent classifiers to reduce variance

Boosting

X Strictly sequential → Longer Traing Time
→ Reweighs training examples → Less Intepretable
→ Averaging dependent classifiers which might be analyzed through dynamic systems and ergodic theory

Extensions : Logit Boost, Grad Boost, L2Boast, XGBoost

Combining Bagging and Boosting :

eg each classifier in Boosting is obtained via Bagging (2000)