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

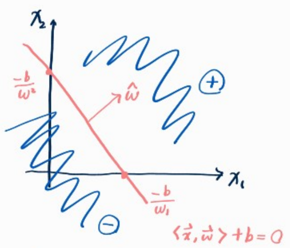
Batch Learning vs Online Learning

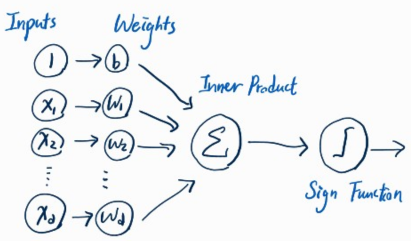
Batch Learning:
- Statistical assumption on training set vs test set

Online Learning:
- streaming data
- interested in making as few mistakes as possible

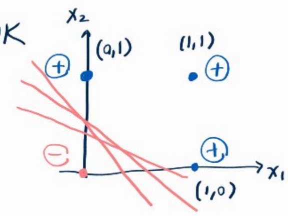
Linear Threshold Function (LFT)

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





OR Dataset → OK



Perceptron Algorithm (1958 by Rosenblatt)

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

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

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

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: $y[\langle \vec{x}, \vec{\omega} \rangle + b] = \langle y\binom{\vec{x}}{1}, \binom{\vec{\omega}}{b} \rangle$, with $\underbrace{y\binom{\vec{x}}{1}}_{\hat{a}}$ (padding) and $\underbrace{\binom{\vec{\omega}}{b}}_{\hat{\omega}}$

Problem becomes: Find $\hat{\omega} \in \mathbb{R}^p$ s.t. $A^T\hat{\omega} > \vec{0}$ (element-wise), 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 $\hat{\omega}$. If each training data is selected infinitely often, then $\forall i$, $\langle y_i \hat{x}_i, \hat{\omega} \rangle > \delta$.

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)

Perceptron vs Support Vector Machine vs Soft-margin Support Vector Machine

* Unique Solution (Support Vector Machine)

* Beyond Separability (Soft-margin Support Vector Machine)

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

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

Least Squares Regression: $\min_{X\to Y} E\|f(X) - Y\|_2^2 = \underbrace{E\|f(X) - m(X)\|_2^2}_{\text{bias}^2} - \underbrace{E\|m(X) - Y\|_2^2}_{\text{noise variance}}$

Appr. Error / Estimation Error

Aim: $f \approx m$ — minimize bias

does not depend on f — inherit measure of the difficulty of the problem

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

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

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

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

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

Apply Iterative Gradient Algorithms !!!!

$\hat{X}$ can be ill-Conditioning e.g. $\hat{X} = \begin{bmatrix} 0 & \varepsilon \\ 1 & 1 \end{bmatrix} \to \hat{\omega} = y\hat{X}^{-1} = [-\frac{2}{\varepsilon}, 1]$ (close to rank deficient) — Slight Perturbation leads to Chaotic behaviour

Ridge Regression

$\min_{\hat{W}} \frac{1}{n}\|\hat{W}\hat{X} - \hat{Y}\|_F^2 + \boxed{\lambda\|\hat{W}\|_F^2}$

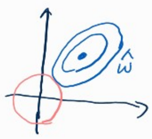
Regularization $\Leftrightarrow$ Constraints $\min_{\|\hat{W}\|_F \le \gamma} \frac{1}{n}\|\hat{W}\hat{X} - \hat{Y}\|_F^2$

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

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

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

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

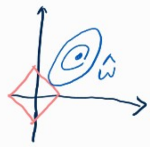


More computation

Lasso Regression (1996)

$\min_{\hat{W}} \frac{1}{n}\|\hat{W}\hat{X} - \hat{Y}\|_F^2 + \boxed{\lambda\|\hat{W}\|_1}$

Regularization $\Leftrightarrow$ Constraints $\min_{\|\hat{W}\|_1 \le \gamma} \frac{1}{n}\|\hat{W}\hat{X} - \hat{Y}\|_F^2$



Task Regularization

$\equiv \min_{\hat{\omega}_\tau} \frac{1}{n}\|\hat{\omega}_\tau\hat{X} - \hat{y}_\tau\|_F^2 + \lambda\|\hat{\omega}_\tau\|_2^2 \quad \forall \tau = 1, \dots, t$

Assumption: Tasks are independent and can be solved separately

⇒ Cross Validation: k-fold validation

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

$\lambda^* = \arg\max_\lambda \text{perf}(\lambda)$

Lec 03 Logistic Regression

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

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: $\Pr(Y_1=y_1, \ldots, Y_n=y_n \mid X_1=\vec{x}_1, \ldots, X_n=\vec{x}_n)$

Maximize this conditional likelihood w.r.t. $\vec{w}$! $= \prod_{i=1}^{n} p(\vec{x}_i;\vec{w}_i)^{\frac{(1+y_i)}{2}} (1-p(\vec{x}_i;\vec{w}_i))^{\frac{(1-y_i)}{2}}$

$\longrightarrow$ figure out $\vec{w}$ $\rightarrow$ 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 $\langle\vec{x},\vec{w}\rangle+b$.

Big Assumption!

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

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

Logistic Loss (after plugging into the Likelihood equation)

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

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

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

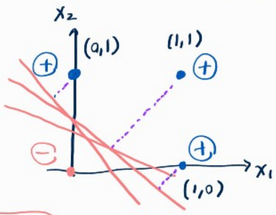
$\hat{p}_i = f(\hat{W}\vec{x}_i) = \text{softmax}(\hat{W}\vec{x})$, softmax: $\mathbb{R}^c \to \Delta_{c-1}$,

$$\eta \mapsto \frac{\exp(\eta)}{\mathbf{1}^T\exp(\eta)}$$

Lec 04 Support Vector Machines

Motivation: Maximizing Margin, ie.

Maximizing the minimum distance to every point of a hyperplane



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

($\hat{y}_i$ above $\langle\vec{x}_i,\vec{w}\rangle+b$)

Quadratic Programming (Perceptron: Linear Programming)　$\min_{\vec{w},b} 0$ s.t. $y_i\hat{y}_i > 0\ \forall i$　Primal form

Unique Solution (Perceptron: infinite solution)

Author: kiking

(vs Lagrangian Dual form)

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 $\forall \alpha \in [0,1]$,

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

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

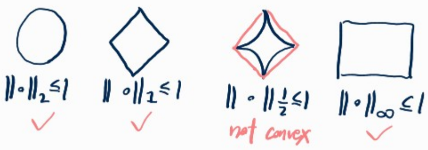
Example: ϕ, $\mathbb{R}^d$ (vector space)

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

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

i.e. the smallest convex superset.

Example: Unit balls of norms
$B_p := \{\|\vec{x}\|_p \le 1\}$ is convex iff $p \geq 1$



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

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

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

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

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

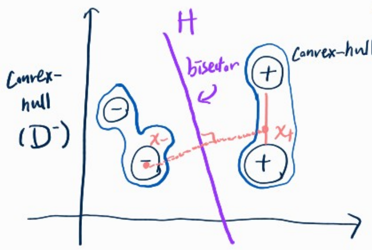
$$conv(A) = \left\{ \vec{x} = \sum_{i=1}^{n} \alpha_i \vec{x}_i : \begin{array}{l} n \in \mathbb{N} \\ \vec{\alpha} \in \Delta_{n-1} \\ \vec{x}_i \in A \end{array} \right\} \quad \alpha_i \geq 0 \ \sum_i \alpha_i = 1$$

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

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

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

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

Author: kiking

w, b $\underbrace{\quad}_{\text{margin maximization}}$ $\sum_{i=1}$ $\underbrace{\quad}_{\text{i-th training error, 0 if } y_i \hat{y}_i \geq 1}$

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

Soft Penalty : the more deviate, the heavier the penalty

Update Rule :

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

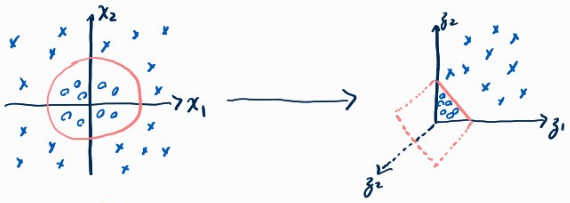
Stochastic Gradient : $\vec{w} \leftarrow \vec{w} - \eta \left[\underbrace{\cancel{}}_{\text{(random sample suffices)}} \ell'(y_I \hat{y}_I) y_I \vec{x}_I + \frac{\vec{w}}{\lambda} \right]$ $O(d)$

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

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

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

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

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

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

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

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

Choose $\phi \longrightarrow$ determines k

determines some $\phi \longleftarrow$ Choose k

decides a unique RKHS (reproducing kernels hilbert space)

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

Verify a Kernel

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

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

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

Examples:

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

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 kernal: $k(s,t) := s \wedge t \quad \forall s,t \geq 0$

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

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

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

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

Training: $O(n^2 d)$

Testing: $O(nd)$

Old

Training: $O(nd)$

Testing: $O(d)$

Trade-Off

✓ avoid explicit dependence on feature dimension (e.g. ∞)

X n times slower in both training + testing

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

Kernel Logistic Regression

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

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

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

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

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 $\vec{w}$

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

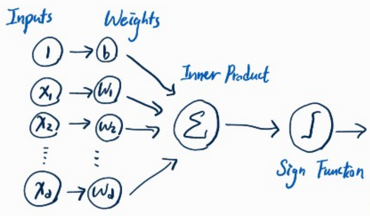
$$\Rightarrow \min_{\vec{w}, \beta} \frac{1}{n} \sum_i \ell\left(\left\langle \bigoplus_p \beta_p \phi_p(\vec{z}_i), \vec{w} \right\rangle\right) + \frac{\lambda}{2} \|\vec{w}\|_2^2$$

Author: kiking

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

Lec 08 Multi-Layer Perceptron

Recall: Perceptron



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

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

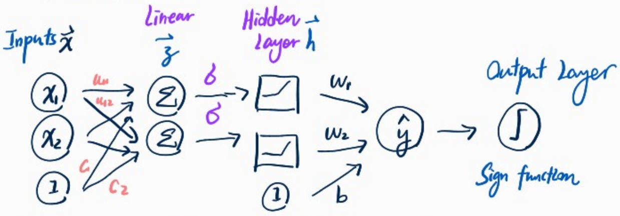
If $y_k \cdot (\langle \vec{x}_k, \vec{w} \rangle + b) \le \xi$ then

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

$b \leftarrow b + y_k$

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



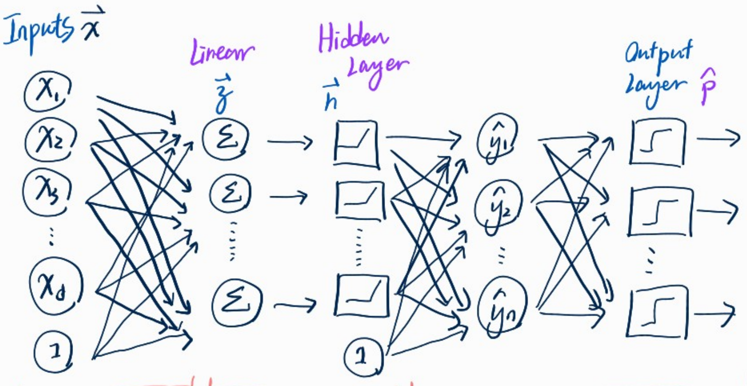
$\underbrace{\vec{z} = U\vec{x} + c}_{\text{1st linear transformation}}$ $\quad U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} \in \mathbb{R}^{2\times 2}$, $c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \in \mathbb{R}^2$

$\underbrace{\vec{h} = \sigma(\vec{z})}$ element-wise non-linear activation

$\underbrace{\hat{y} = \langle \vec{h}, \vec{w} \rangle + b}$ 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) $\Downarrow$ 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

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

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

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

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

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, mini batch $B \subseteq \{1,\dots,n\}$ suffices

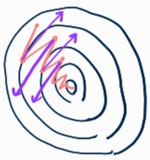
$$w \leftarrow w - \eta \cdot \underbrace{\frac{1}{|B|}\sum_{i\in B}}_{\text{instead of } \frac{1}{n}\sum_{i=1}^{n} \text{ (full pass over the training set)}} \nabla\left[\ell(f(\vec{x}_i;\vec{w}), y_i)\right]$$

Trade-off: variance ↑↑
computation ↓↓

Momentum

Intepretation: gradient averaging

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



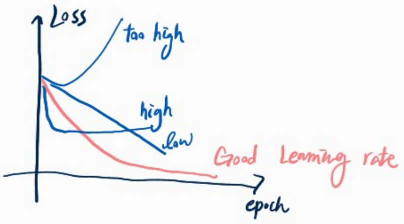
Nesterov Momentum: averaging gradients looking ahead

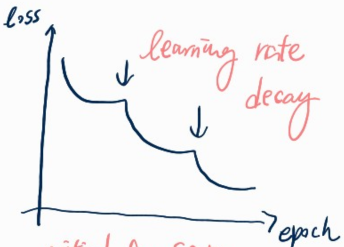
$$\vec{w}_{t+1} = \vec{z}_t - \eta_t \nabla g(\vec{w}_t), \quad \vec{z}_{t+1} = \underbrace{\vec{w}_{t+1} + \beta_t(\vec{w}_{t+1} - \vec{w}_t)}_{\text{looks ahead}}$$

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





critical for SGD+ momentum (less for Adam)

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

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

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

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: $\underbrace{\text{conv} \to \text{relu} \to \text{conv} \to \text{relu} \to \text{pool}}_{\text{repeat}} \to \underbrace{\text{FC} \to \text{relu}}_{\text{repeat}} \to \text{FC}$

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:

$h^{(l)} \to \text{Conv} \to \text{Relu} \to \text{Conv} \to \text{ReLu} \to h^{(l+2)}$

a residual block

Author: kiking

Short-cut connection that allows "skipping" one or more layers

Allow more direct backpropogation of the gradient

Examples:

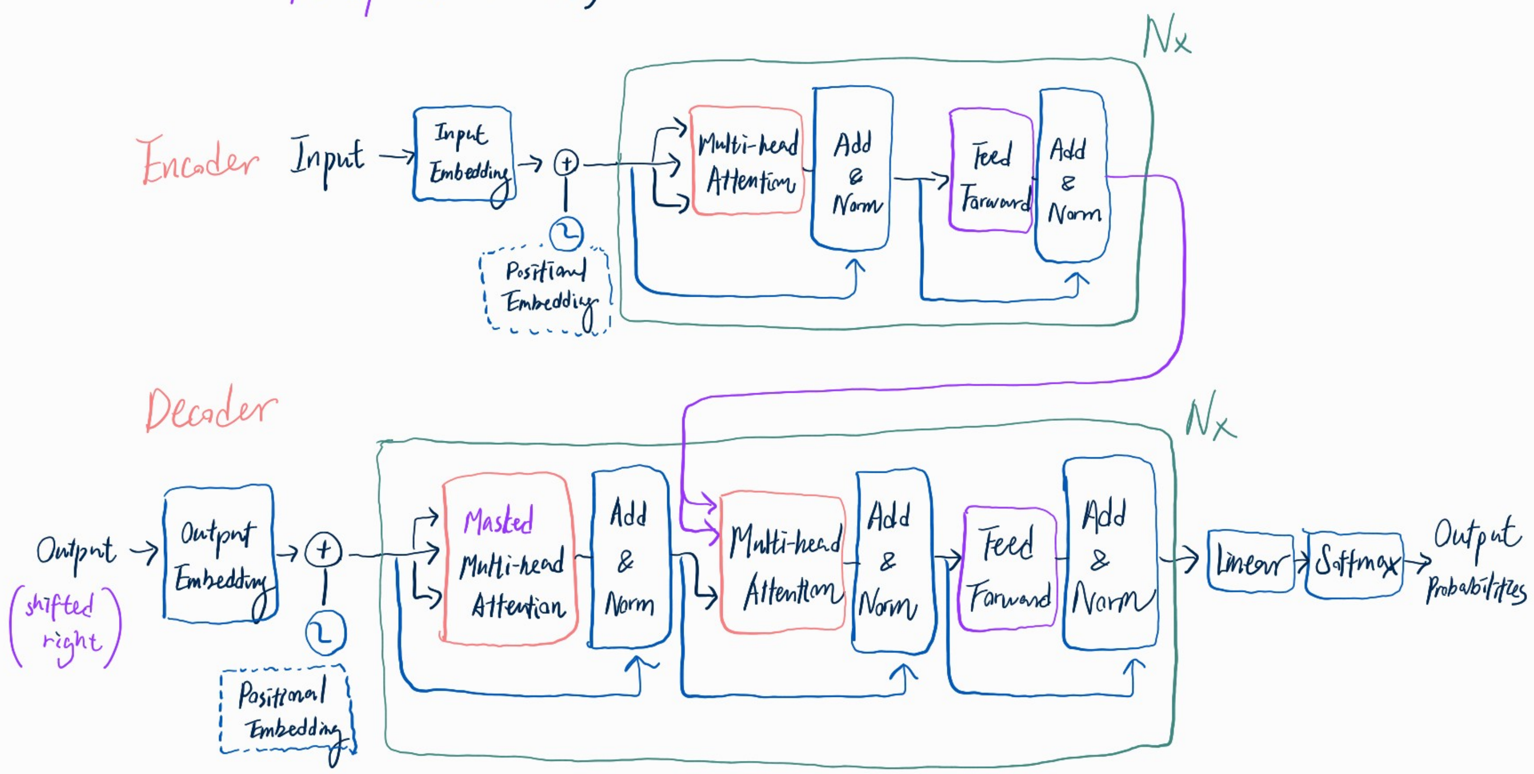
ResNet

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 $\begin{pmatrix} \vec{y}_1 \\ \vec{y}_2 \\ \vdots \\ \vec{y}_\ell \end{pmatrix}$ $\vec{x}_m, \vec{y}_\ell \in \mathbb{R}^p$ one-hot

p: dictionary size

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

$$\overbrace{\begin{pmatrix} \vec{x}_1 \\ \vdots \\ \vec{x}_m \end{pmatrix}}^{p} \cdot \; p\left\{ \overbrace{\begin{pmatrix} \vec{w}_1 & \vec{w}_2 & \cdots & \vec{w}_d \end{pmatrix}}^{d} \right.$$

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

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

$$\begin{array}{c} \overbrace{\qquad\qquad}^{d} \\ t \begin{pmatrix} \vdots & \vdots \\ \cdots \circ & \circ \cdots \\ \vdots & \vdots \end{pmatrix} \\ 2i \quad 2i{+}1 \end{array} \qquad w^p_{2i} = \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right) \quad w^p_{2i+1} = \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right)$$

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

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):

$$K = [\vec{k}_1, \dots, \vec{k}_m]^T \in \mathbb{R}^{m \times d}, \quad V = [\vec{k}_1, \dots, \vec{k}_m]^T \in \mathbb{R}^{m \times d}$$

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):

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

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

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 $\lambda = \sqrt{d_k} \rightarrow \vec{q} \perp \vec{k} \sim N(0, I_{d_k})$, $\text{Var}\left(\frac{\vec{q}^T \vec{k}}{\sqrt{d_k}}\right) = 1$
- 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:

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

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

Self-Attention

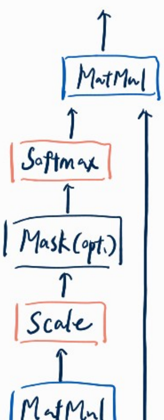
$$V \leftarrow VW^v$$

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

- 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

$\boxed{\text{att}(QW^q, KW^k, VW^v)}$



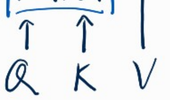
$$[Q, K, V] \leftarrow [QW^q, KW^k, VW^v]$$

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

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

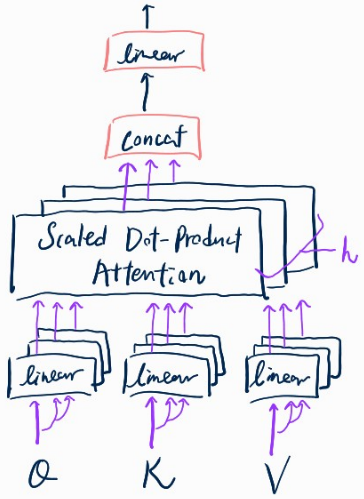
- Dot product between Query Q and Key K: measures similarity
- Output: convex combination of input
- Self-Attention: a special case

Author: kiking



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

Multi-head Attention



$h = 8$ or 12

(keep $d = 512$)

for $i=1,\dots,h$ do $\boxed{V_i = att(QW_i^q, KW_i^k, VW_i^v)}$

// linear transformation

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

// convex combination

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

// 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

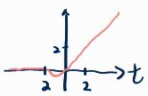
$FFN(\hat{V}) = \sigma(\hat{V}\hat{W}_1)\hat{W}_2$ → Two-layer Network

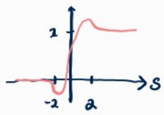
$\hat{V} \in \mathbb{R}^{len\times d}$ → values at some layer, arranged row-wise

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

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

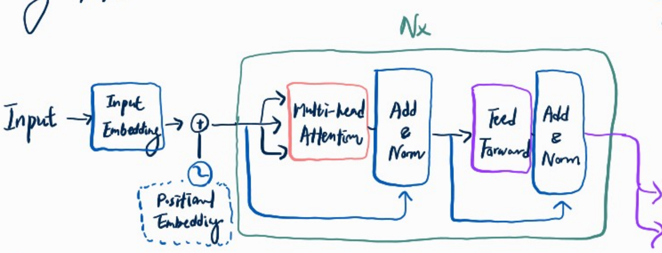
$y(t) = t\cdot\Phi(t)$ $\qquad$ $y'(t) = t\phi(t) + \Phi(t)$





Encoder: Understanding Attention

Encoder



for Decoder (context K,V)

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

Self-Attention

$V \leftarrow VW^v$

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

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

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

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

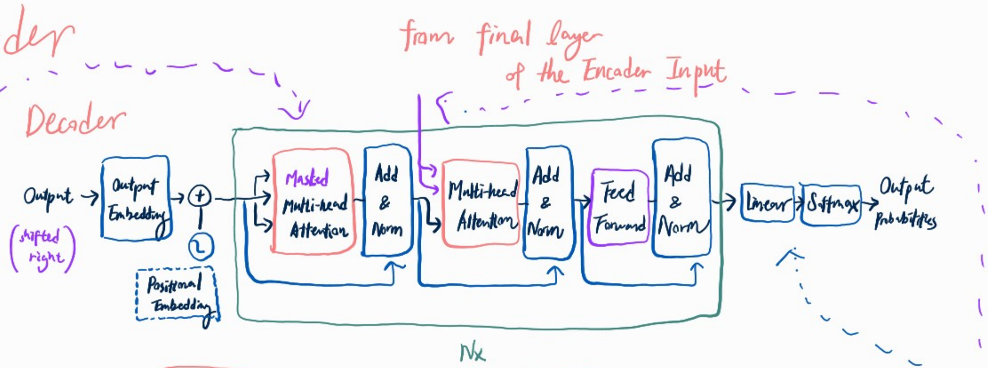
③ 1×1 convolution with $4d$ filters
$W = [\vec{w}_1,\dots,\vec{w}_{4d}]$ s.t. $u_{ik} \leftarrow \langle \vec{v}_i, \vec{w}_k\rangle$

④ Activation with GELU

⑤ 1×1 convolution with d filters again

Author: kiking

Decoder



$$[Q, K, V] \longleftarrow [VW^q, VW^k, VW^v]$$

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

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

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

minimize the multi-class cross-entropy loss

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

$\hat{Y} = [\hat{y}_1, \ldots, \hat{y}_\ell]$ 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 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	(per-layer) Complexity	Sequential Operation	Max. Path Length
Self-Attention	$O(m^2 d)$ √parallizable	$O(1)$	$O(1)$
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

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

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: ↑↑ max. path length to $O(\frac{m}{r})$

Suitable for modelling long-range dependencies

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)

$$\underbrace{\vec{y}}_{\text{original label vector (typical one-hot)}} \leftarrow \underbrace{(1-\alpha)}_{\alpha:\text{ controls the amount of smoothing}} \vec{y} + \alpha \underbrace{\frac{1}{C}}_{C:\text{ number of classes}}$$

Beam Search:

Example: Machine Translation

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

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

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

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: Contexualized 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 $\vec{x}$:

$$\text{ELMo}(\vec{x}; A) = \underbrace{A}_{\text{Aggregation function}}(\underbrace{h_0}_{\text{embedding shared between the two L-layer LSTMs}}, \underbrace{\{\overrightarrow{h}_\ell\}_{\ell=1}^{L}, \{\overleftarrow{h}_\ell\}_{\ell=1}^{L}}_{\overrightarrow{h}, \overleftarrow{h} = \text{hidden states of LSTMs}})$$

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

Total 3 layers of representation: ① context-insensitive through character-level CNN
② forward & backward & ③ L-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 Representation from Transformers

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

Input: Concatenation of two sequences:
[CLS] sequence A [SEP] sequence B [SEP]

Final Representation:
Sentence-level abtraction 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

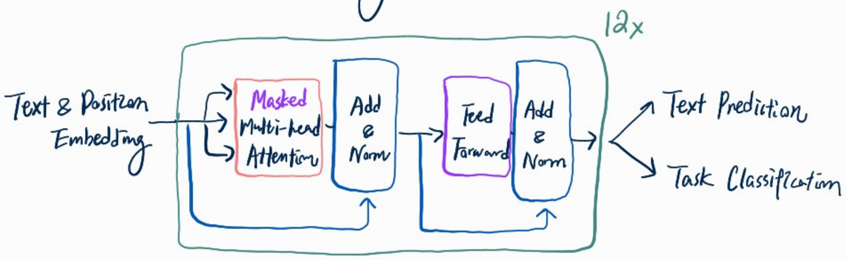
Fine-tuning (FT):

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-Traing



→ Unsupervised pre-training through a LM:

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

given context of previous tokens, predict current token

probabilities modelled by Multi-Layer Transformer Decoder

$H^{(0)} = \underbrace{X}\underbrace{W^e} + \underbrace{W^p}$ $\left(\begin{smallmatrix}\vec{x}_1 \\ \vdots \\ \vec{x}_m\end{smallmatrix}\right)$ input sequence of m tokens (vary length) | token embedding matrix | position embedding matrix

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

Author: kiking

$\ell = 1, \dots, L$; number of transformer blocks

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

→ 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)

$\sigma(x) = x\Phi(x) = \mathbb{E}(x \cdot m | x)$ 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
converges after 3 epoches
$\lambda = \frac{1}{2}$

(learning rate)
warm-up $\tau = 2000$ during pre-training
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
involves task-dedicated datasets and [gradient updates on both the LM] and the task-dependent classifier

never for GPT-3

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) $\xrightarrow{\text{Alignment}}$ 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 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