CS480: Introduction to Machine Learning

Felix Zhou [1]

June 1, 2023

[1]From Professor Yaoliang Yu's Lectures at the University of Waterloo in Fall 2021

Contents

Part I

Background

Chapter 1

Optimization

1.1 Definitions

Definition 1.1.1 (Eigenvalue)
A scalar $\lambda \in \mathbb{C}$ and a vector $v \in \mathbb{C}^d$ are an eigenvalue and eigenvector of a square matrix $A \in \mathbb{R}^{d \times d}$ if

$$Av = \lambda v.$$

Recall that every matrix has d eigenvalues. For real symmetric matrices, eigenvalues and eigenvectors are real. Moreover, there are d orthogonal eigenvectors.

Definition 1.1.2 (Singular Value)
The singular values of a matrix A are the square root of eigenvalues of AA^T.

Note that AA^T is symmetric.

Definition 1.1.3 (Dual Norm)
The dual norm $\|\cdot\|_\circ$ of the norm $\|\cdot\|$ is defined as

$$\begin{aligned}\|z\|_\circ &:= \max_{\|w\|=1} w^T z \\ &= \max_{w\neq 0} \frac{w^T z}{\|w\|} \\ &= \max_{\|w\|=1} |w^T z| \\ &= \max_{w\neq 0} \frac{|w^T z|}{\|w\|}\end{aligned}$$

From definition, we have

$$w^T z \leq |w^T z| \leq \|w\| \cdot \|z\|_\circ.$$

The dual norm of the ℓ_p norm is the ℓ_q norm where q is the Hölder conjugate of p. Hence the Cauchy-Schwarz inequality can be explained by the self-duality property of the ℓ_2 norm:

$$w^T z \leq |w^T z| \leq \|w\|_2 \cdot \|z\|_2.$$

For any $1 \leq q \leq p \leq \infty$,

$$\|w\|_p \leq \|w\|_q \leq d^{\frac{1}{q}-\frac{1}{p}} \|w\|_p.$$

Hence all ℓ_p norms are equivalent. In fact, all norms on finite dimensional vector spaces are equivalent.

Definition 1.1.4 (Frobenius Norm)
For $A \in \mathbb{R}^{n\times d}$, its Frobenius norm is defined as

$$\|A\|_F := \sqrt{\sum_{ij} a_{ij}^2}.$$

Recall the operator norm. For a matrix $A : (\mathbb{R}^d, \|\cdot\|_2) \to (\mathbb{R}^n : \|\cdot\|_2)$, the operator norm coincides with the spectral norm $\|\cdot\|_{\text{Sp}}$, which is also the largest singular value of A.

It is known that

$$\|A\|_{\text{Sp}} \leq \|A\|_F \leq \sqrt{\text{rank } A}\|A\|_{\text{Sp}}.$$

Definition 1.1.5 (Convex Function)
An extended real-valued function $f : V \to (-\infty, \infty]$ is convex if Jensen's inequality holds:

$$f(\lambda w + (1-\lambda)z) \leq \lambda f(w) - (1-\lambda)f(z)$$

for all $w, z \in V$ and $\lambda \in (0,1)$.

If $f : \mathbb{R}^d \to \mathbb{R}$ is convex, then so is $w \mapsto f(Aw + b)$.

The supremum of a collection of convex functions is convex.

If $f(w,t)$ is jointly convex in w, t, then $w \mapsto \inf_t f(w,t)$ is convex.

If $f : C \to \mathbb{R}$ is convex, then the perspective function

$$g(w,t) := tf(w/t)$$

is convex on $C \times \mathbb{R}_{++}$.

Definition 1.1.6 (Fenchel Conjugate)
The Fenchel conjugate function of an extended real-valued function $f : V \to (-\infty, \infty]$ is defined as

$$f^*(w^*) := \sup_w \langle w, w^* \rangle - f(w).$$

f^* is always convex and lower semi-continuous.

$f = f^{**}$ if and only if f is convex and lower semi-continuous.

Theorem 1.1.1
Let $f : \mathbb{R}^d \to \mathbb{R}$ be differentiable.

(i) f is convex if and only if for all w, z,

$$f(z) \geq f(w) + \langle z - w, \nabla f(w) \rangle.$$

(ii) If f is twice differentiable, then f is convex if and only if for all w, $\nabla^2 f(w) \succeq 0$.

(iii) If $\nabla^2 f(w) \succeq 0$ for all w, then f is strictly convex.

Definition 1.1.7 (Level-Bounded)
A function $f : \mathbb{R}^d \to (-\infty, \infty]$ is level bounded if and only if for all $t \in \mathbb{R}$, the sublevel set $[\![f \leq t]\!]$ is bounded.

Equivalently, f is level bounded if and only if $\|w\| \to \infty \implies f(w) \to \infty$.

1.2 Optimization

Theorem 1.2.1
A continuous and level-bounded function $f : \mathbb{R}^d \to (-\infty, \infty]$ has a minimizer.

Theorem 1.2.2
Any local minimizer of a convex function is global.

Theorem 1.2.3
A necessary condition to be a local minimizer of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is that

$$\nabla f(w) = 0.$$

If f is convex, this is also sufficient.

We refer to such points as *stationary/critical points*.

1.2.1 Algorithm of Feasible Direction

We wish to solve

$$\begin{aligned} &\max f(x) \\ &x \in C \end{aligned}$$

This class of algorithms begin with a feasible solution w_t, finds a normalized direction vector d_t, then updates the solution

$$w_{t+1} := w_t - \eta_t d_t.$$

Here $\eta_t > 0$ is the step size.

In machine learning, we typically minimize some average loss over a training set $\mathcal{D} := \{(x_i, y_i)\}$

$$\min_w \sum_{i=1}^{n} \ell(w; x_i, y_i) = \min_w E[\ell(w; x, y)]$$

where (x, y) is randomly chosen from the training set $\mathcal{D}$ and the empirical expectation is ratken with respect to $\mathcal{D}$. Computing the gradient is costly, but by taking a random sample $(x, y) \in \mathcal{D}$ and taking

$$\xi = \nabla \ell(w; x, y),$$

$E[\xi]$ is the equal to the gradient but comes at a much cheaper computational cost. In practice, we sample a mini-batch and compute the average gradient over the mini-batch.

Choosing a Direction

Here are some ways to choose the direction

Gradient Descent (GD) $d_t = \nabla f(w_t)$

Newton $d_t = [\nabla^2 f(w_t)]^{-1} \nabla f(w_t)$

Stochastic Gradient Descent (SGD) $d_t = \xi_t$ where ξ_t is a random variable satisfying $E[\xi_t] = \nabla f(w_t)$

Choosing a Step Size

Here are a few ways to help choose η_t

Cauchy's Rule $\eta_t \in \operatorname{argmin}_{\eta \geq 0} f(w_t - \eta d_t)$, where the existence of a minimizer is assumed

Curry's Rule $\eta_t = \inf\{\eta \geq 0 : f'(w_t - \eta d_t) = 0\}$, where the finiteness of η_t is assumed

Constant Rule $\eta_t \equiv \eta > 0$

Summable Rule $\sum_t \eta_t = \infty, \sum_t \eta_t^2 < \infty$, with $\eta_t = O(1/t)$ as an example

Diminishing Rule $\sum_t \eta_t = \infty, \lim_t \eta_t = 0$, with $\eta_t = O(1/\sqrt{t})$ as an example

The latter 3 rules are most common, especially for SGD.

Curry's rule leads to larger per-step decrease of the function value but is also computationally more demanding. Under both Cauchy and Curry's rule, we have the orthogonality property:

$$\langle d_t, \nabla f(w_{t+1}) \rangle = 0.$$

this explains the zigzag behavior in some gradient algorithms.

1.2.2 Lagrangian Dual

Consider the optimization problem

$$\begin{aligned} &\min_{w \in C \subseteq \mathbb{R}^d} f(w) \\ g(w) &\leq 0 \\ h(w) &= 0 \end{aligned}$$

The constraint set C represent the "simple" cosntraints. We introduce Lagrangian multipliers (dual variables) to handle constraints g, h

$$L(w; \mu, \nu) := f(w) + \mu^T g(w) + \nu^T h(w).$$

The original problem then becomes the min-max

$$\inf_{w \in C} \sup_{\mu \geq 0, \nu} L(w; \mu, \nu). \qquad (P^*)$$

For any infeasible solution to the original problem, we can find some μ, ν which makes $L(w; \mu, \nu) = \infty$, hence the minimizer must be a feasible solution to the original problem.

The Lagrangian dual swaps the order of the min max

$$\sup_{\mu \geq 0, \nu} \inf_{w \in C} L(w; \mu, \nu). \qquad (D^*)$$

Theorem 1.2.4
For any function $f : W \times Z \to \mathbb{R}$,

$$\inf_w \sup_z f(w, z) \geq \sup_z \inf_w f(w, z).$$

When equality holds, we say *strong duality* holds. For example, if for all $i \in [n]$, f, g_i, and C are convex, h is affine, and some "mild regularity conditions" (Slater's condition) holds, then we have strong duality for the Lagrangian.

We can apply our algorithm above to solve the Lagrangian dual. The with an optimal dual variable μ^*, ν^*, we "recover" the primal solution

$$\mathfrak{M}(\mu^*, \nu^*) := \text{argmin}_{w \in C} L(w; \mu^*, \nu^*).$$

$\mathfrak{M}$ always contains all minimizers of the primal problem. However, it may be a strict superset! When strong duality holds, we can verify primal feasibility in order to identify the true minimizers of the primal. Unfortunately, our algorithm only returns one solution from $\mathfrak{M}$ in practice. Thus if it is not primal feasible, we may not be able to fetch a different solution from $\mathfrak{M}$.

1.2.3 Gradient Descent Ascent

The Lagrangian dual is especially useful when we can solve the inner optimization problem analytically. When that is not possible, an alternative is to perform one gradient descent step on the inner minimization and then perform another gradient ascent step on the outer maximization.

In general, consider the problem

$$\inf_{w \in W} \sup_{z \in Z} f(w, z).$$

The input is some $(w_0, z_0) \in \operatorname{dom} f \cap (X \times Z)$. For $t = 0, 1, \ldots$:

1) choose step size $\eta_t > 0$
2) $w_{t+1} := P_W(w_t - \eta_t \boldsymbol{\nabla}_w f(w_t, z_t))$
3) $z_{t+1} := P_Z(z_t + \eta_t \boldsymbol{\nabla}_z f(w_t, z_t))$
4) $s_t := s_{t-1} + \eta_t$
5) $(\bar{w}_t, \bar{z}_t) := \frac{s_{t-1}(\bar{w}_{t-1}, \bar{z}_{t-1}) + \eta_t(w_t, z_t)}{s_t}$

The last two steps are an optional weighted averaging step

$$(\bar{w}_t, \bar{z}_t) := \sum_{k=1}^{t} \eta_k (w_k, z_k) / \sum_k \eta_k$$

Variations of GDA include using different step sizes on w, z, using w_{t+1} in the update on z or vice versa, using stochastic gradients in both steps, and performing k updates in w after every update in w or vice versa.

It is known that vanilla GDA may NEVER converge for any step size. Convergence may be recovered through averaging but the rate of convergence can be slow.

1.2.4 Alternating Algorithm

Consider the joint minimization problem

$$\inf_{w \in W} \inf_{z \in Z} f(w, z).$$

The following algorithm is often applied in practice.

The input is some $(w_0, z_0) \in \operatorname{dom} f \cap (W \times Z)$. For $t = 0, 1, \ldots$:

1. $w_{t+1} := \operatorname{argmin}_{w \in W} f(w, z_t)$
2. $z_{t+1} := \operatorname{argmin}_{z \in Z} f(w_{t+1}, z)$

Note that we perform an exact minimization for each step.

It can be tempting to adapt this algorithm for min-max problems, but being more aggresive here can actually hurt and we may never converge. Once again, averaging may recover convergence.

Chapter 2

Statistical Learning

2.1 Definitions

Recall that the *cumulative distribution function (cdf)* of a random vector $X \in \mathbb{R}^d$ is

$$F(x) := P(X \leq x).$$

Its *probability density function (pdf)* is

$$p(x) := \frac{\partial^d F}{\partial x_1 \dots \partial x_d}(x).$$

Equivalently,

$$F(x) = \int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_d} p(x) dx.$$

Each cdf is monotonically increasing in each input, right continuous in each input, and $\lim_{x \to \infty} F(x) = 1$ and $\lim_{x \to -\infty} F(x) = 0$.

Each pdf integrates to 1, ie

$$\int_{-\infty}^{\infty} p(x) dx = 1.$$

Theorem 2.1.1 (Change of Variable)
Let $T : \mathbb{R}^d \to \mathbb{R}^d$ be a diffeomorphism (differentiable bijection with differentiable inverse) and Z a random vector.
Let $X := T(Z)$. Then

$$p(x) = q(T^{-1}(x))\left|\det \frac{dT^{-1}}{dx}(x)\right|$$
$$q(z) = p(T(z))\left|\det \frac{dT}{dz}(z)\right|$$

Let $X = (X_1, X_2)$ be a random vector with pdf $p(x) = p(x_1, x_2)$. We say X_1 is a *marginal* of X with pdf

$$p_1(x_1) = \int_{-\infty}^{\infty} p(x_1, x_2)dx_2.$$

Similarly, X_2 is a marginal of X.

We define the *conditional* $X_1 \mid X_2$ with density

$$p_{1|2}(x_1 \mid x_2) := p(x_1, x_2)/p_2(x_2).$$

The value of $p_{1|2}$ is arbitrary if $p_2(x_2)$ (usually does not matter). Similarly, we can define the conditional $X_2 \mid X_1$.

The joint density p can be factorized into the product of marginal $p_1, p_{2|1}$:

$$p(x_1, x_2) = p_1(x_1)p_{2|1}(x_2 \mid x_1) = p_2(x_2)p_{1|2}(x_1 \mid x_2).$$

More generally, the chain rule states that

$$p(x_1, \ldots, x_d) = \prod_{j=1}^{d} p(x_j \mid x_1, \ldots, x_{j-1}).$$

We say the random vectors $X_1, \ldots, X_d$ are *independent* if

$$p(x_1, \ldots, x_d) = \prod_{j=1}^{d} p(x_j).$$

All constructions above can be expressed through cdfs at greater complication. In particular, Baye's rule states that

$$P(A \mid B) = \frac{P(A, B)}{P(B)} = \frac{P(B \mid A)P(A)}{P(B, A) + P(B, \neg A)}.$$

Let $X = (X_1, \ldots, X_d)$ be a random vector. We define its mean as $\mu = \mathbb{E}[X]$ where

$$\mu_j := \int x_j \cdot p(x_j) dx_j.$$

and its covariance matrix as $\Sigma = \mathbb{E}[(X - \mu)(X - \mu)^T]$. In other words,

$$\Sigma_{ij} = \int (x_i - \mu_i)(x_j - \mu_j) \cdot p(x_i, x_j) dx_i dx_j.$$

The j-th diagonal entry of the covariance $\sigma_j^2 := \Sigma_{jj}$ is the *variance* of X_j.

Proposition 2.1.2
We can alternative express the covariance as

$$\Sigma = \mathbb{E}[XX^T] - \mu\mu^T$$
$$\Sigma = \frac{1}{2}\mathbb{E}[(X - X')(X - X')^T]$$

where X' is iid with X.

Definition 2.1.1 (Multivariate Gaussian)
The pdf of the d-dimensional Gaussian (normal) distribution is

$$p(x) = (2\pi)^{-d/2}[\det \Sigma]^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right).$$

An important and characterizing property of the normal distribution is equivariance under affine transformations:

$$X \sim N(\mu, \Sigma) \implies AX + b \sim N(A\mu + b, A\sigma A^T).$$

2.2 Bias-Variance Trade-Off

We wish to predict a random scalar Y based on some feature vector X using the (random) function $\hat{f}$. We often use the squared loss to evaluate predictions:

$$\begin{aligned}\mathbb{E}[(\hat{f}(X) - Y)^2] &= E\left[\left(\hat{f}(X) - \mathbb{E}[\hat{f}(X)] + \mathbb{E}[\hat{f}(X)] - \mathbb{E}[Y \mid X] + \mathbb{E}[Y \mid X] - Y\right)^2\right] \\ &= \underbrace{E\left[\left(\hat{f}(X) - \mathbb{E}[\hat{f}(X)]\right)^2\right]}_{\text{variance}} + \underbrace{E\left[\left(\mathbb{E}[\hat{f}(X)] - \mathbb{E}[Y \mid X]\right)^2\right]}_{\text{bias}^2} + \underbrace{E\left[(\mathbb{E}[Y \mid X] - Y)^2\right]}_{\text{difficulty}}\end{aligned}$$

Note the inherent difficulty of predicting Y with data X. It is independent of $\hat{f}$.

The variance reflects the fluctuation incurred by training on some random training set. A less flexible $\hat{f}$ incurs a smaller variance.

The squared bias term reflects the mismatch of our choice of $\hat{f}$ and the optimal regression function. A very flexible $\hat{f}$ incurs a smaller bias.

One major goal of machine learning is to strike an appropriate balance between the two terms.

2.3 Maximum Likelihood Estimation

Suppose we have a dataset $\mathcal{D} = \{x_i : i \in [n]\}$, where each sample x_i follows some pdf $p(x \mid \theta)$ with unknown parameter θ.

The *likelihood* of a parameter θ given $\mathcal{D}$ is

$$\begin{aligned} L(\theta) &= L(\theta; \mathcal{D}) \\ &:= p(\mathcal{D} \mid \theta) \\ &= \prod_{i=1}^{n} p(x_i \mid \theta) && \text{assuming independence} \end{aligned}$$

A popular way to estimate θ is to maximize the likelihood over some parameter space Θ,

$$\tilde{\theta} := \text{argmax}_{\theta \in \Theta} L(\theta).$$

Equivalently, by taking the log and negating, we minimize the negative *log-likelihood*

$$\tilde{\theta} := \text{argmin}_{\theta \in \Theta} \sum_{i=1}^{n} -\log p(x_i \mid \theta).$$

MLE is only applicable when the likelihood function can be evaluated efficiently!

> **Example 2.3.1**
> We can phrase ordinary linear regression as MLE!

2.3.1 Sample Mean & Covariance as MLE

Let $x_1, \ldots, x_n$ be iid samples from the normal distribution $N(\mu, \Sigma)$ with unknown μ, Σ. It can be shown that

$$\begin{aligned}\hat{\mu}_{\text{MLE}} &:= \text{argmin}_\mu \frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\\ &= \frac{1}{n}\sum_{i=1}^{n}x_i.\end{aligned}$$

Similarly, we can show that

$$\begin{aligned}\hat{\Sigma}_{\text{MLE}} &:= \text{argmin}_\Sigma \log\det\Sigma + \sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\\ &= \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T.\end{aligned}$$

If μ is unknown, we can use $\hat{\mu}_{\text{MLE}}$.

2.4 f-Divergence

Definition 2.4.1 (f-Divergence)
Let $f : \mathbb{R}_+ \to \mathbb{R}$ be a strictly convex function with $f(1) = 0$. The f-divergence measures the similarity between pdfs p, q:

$$D_f(p\|q) := \int f(p(x)/q(x)) \cdot q(x)dx.$$

Here as assume that $q(x) = 0$ implies that $p(x) = 0$. Otherwise, $D_f(p\|q) = \infty$.

We define

$$f^\diamond(t) := t \cdot f(1/t).$$

Proposition 2.4.1
Let D_f be some f-divergence.

- $D_f(p\|q) \geq 0$ with equality attained if and only if $p = q$
- $D_{f+g} = D_f + D_g$
- For $s > 0$, $D_{sf} = sD_f$
- If $g(t) := f(t) + s(t-1)$ for some $s \in \mathbb{R}$, then $D_g = D_f$
- If p, q have the same support, then $D_f(p\|q) = D_{f^\diamond}(q\|p)$
- $f^\diamond$ is strictly convex with $f^\diamond(1) = 0$ and $(f^\diamond)^\diamond = f$

f-divergences are not usually symmetric. However, we can always symmetrize them by the transformation $f' := f + f^\diamond$.

Definition 2.4.2 (Killback-Leibler Divergence)
Let $f(t) := t \log t$. We obtain the KL divergence

$$\mathrm{KL}(p\|q) = \int p(x) \log(p(x)/q(x)) dx.$$

Reversing the inputs yield the reverse KL-divergence

$$\mathrm{KL}(p\|q) := \mathrm{KL}(q\|p).$$

The underlying function for reverse KL is $f(t) = -\log t$.

2.5 Information Theory

Definition 2.5.1 (Entropy)
The entropy of a random vector X with pdf p is

$$H(X) := \mathbb{E}[-\log p(X)] = -\int p(x) \log p(x) dx.$$

We can understand this as the expected uncertainty in X. $H(X)$ is approximately equal to how much "information" we learn on average from one instance of the random variable X.

Definition 2.5.2 (Conditional Entropy)
The conditional entropy between X, Z with pdfs p, q is

$$\begin{aligned} H(X \mid Z) &:= \mathbb{E}_{X,Y}[-\log p(X \mid Z)] \\ &= -\int p(x,z) \log p(x \mid z) dx dz \\ &= -\int p(x,z) \log \frac{p(x,z)}{p(z)} dx dz \\ &= -\int_z q(z) \int_x p_{X|Z}(x \mid z) \log p(x \mid z) dx dz \\ &= \int_z q(z) H(X \mid Z = z) dz. \end{aligned}$$

The conditional entropy is a measure of how much uncertainty remains about X when we know the value of Z.

Definition 2.5.3 (Cross-Entropy)
The cross-entropy between X, Z with pdfs p, q is

$$\dagger(X, Z) := \mathbb{E}[-\log q(X)] = -\int p(x) \log q(x) dx.$$

In some sense, this is a measure of difference between two distributions.

Definition 2.5.4 (Mutual Information)
The mutual information between X, Z is

$$I(X, Z) := \mathrm{KL}(p(x,z) \| p(x)q(z)) = \int p(x,z) \log \frac{p(x,z)}{p(x)q(z)} dx dz.$$

Mutual information tells us how much we learn about X from knowing the value of Y.

Proposition 2.5.1
The following hold:

(i) $H(X, Z) = H(Z) + H(X \mid Z)$

(ii) $\dagger(X, Z) = H(X) + \mathrm{KL}(X \| Z) = H(X) + \mathrm{LK}(Z \| X)$

(iii) $I(X, Z) = H(X) - H(X \mid Z)$

(iv) $I(X, Z) \geq 0$ with equality if and only if X is independent of Z

(v) $\mathrm{KL}(p(x,z) \| q(x,z)) = \mathrm{KL}(p(z) \| q(z)) + \mathbb{E}[\mathrm{KL}(p(x \mid z) \| q(x \mid z))]$

Example 2.5.2 (MLE as KL Minimization)
Consider the empirical pdf based on a dataset $\mathcal{D} := \{x_1, \ldots, x_n\}$

$$\hat{p}(x) = \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}.$$

Then

$$\begin{aligned}\mathrm{KL}(\hat{p}\|p(x \mid \theta)) &= \int [\log \hat{p}(x) - \log p(x \mid \theta)]\hat{p}(x)dx \\ &= C + \frac{1}{n}\sum_{i=1}^{n} -\log p(x_i \mid \theta)\end{aligned}$$

where C is a universal constant.

Thus $\theta_{\mathrm{MLE}} = \mathrm{argmin}_{\theta\in\Theta} \mathrm{KL}(\hat{p}\|p(x \mid \theta))$!

2.6 Bayesian Statistics

Definition 2.6.1 (Prior)
In a full Bayesian approach to parameter estimation, we assume the parameter θ is random and follows a prior pdf $p(\theta)$.

Ideally, we choose $p(\theta)$ to encode a priori knowledge of the problem at hand. In practice, computational convenience is often prioritized instead.

Definition 2.6.2 (Posterior)
Suppose we have a prior pdf $p(\theta)$. After observing some data $\mathcal{D}$, our belief on the probable values of θ have changed and we obtain the posterior:

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{p(\mathcal{D})} = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{\int p(\mathcal{D} \mid \theta)p(\theta)d\theta}.$$

Computing the denominator may be difficult since it involves an integral that may not be tractable.

Example 2.6.1
Consider OLS $Y = Xw + \epsilon$ where $\epsilon \sim N(\mu, S)$ and we impose a Gaussian prior on the weights $w \sim N(\mu_0, S_0)$. We also assume ϵ, w are independent.

Given a dataset $\mathcal{D} = \{(X_1, y_1), \ldots, (X_n, y_n)\}$, we can compute the posterior

$$p(w \mid \mathcal{D}) \propto p(w)p(\mathcal{D} \mid w) = N(\mu_n, S_n).$$

By completing the square,

$$\begin{aligned} S_n^{-1} &= S_0^{-1} + \sum_{i=1}^{n} X_i^T S^{-1} X_i \\ \mu_n &= S_n \left(S_0^{-1}\mu_0 + \sum_{i=1}^{n} X_i^T S^{-1}(y_i - \mu) \right). \end{aligned}$$

We can also express ordinary linear regression using Bayesian statistics.

Theorem 2.6.2 (Bayes Classifier)
Consider the classification problem with random variables $X \in \mathbb{R}^d$ and $Y \in [c]$. The optimal (Bayes) classification rule define as

$$\text{argmin}_{h:\mathbb{R}^d \to [c]} P(Y \neq h(X))$$

admits the closed-form formula

$$\begin{aligned} h^\star(x) &= \text{argmax}_{k \in [c]} P(Y = k \mid X = x) \\ &= \text{argmin}_{k \in [c]} \underbrace{p(X = x \mid Y = k)}_{\text{likelihood}} \cdot \underbrace{P(Y = k)}_{\text{prior}}. \end{aligned}$$

Ties can be broken arbitrarily.

The *Bayes error* achieved by the Bayes classification rule is

$$\mathbb{E}[1 - \max_{k \in [c]} P(Y = k \mid X)].$$

2.7 Maximum A Posteriori

Another popular parameter estimation algorithm is MAP which maximizes the posterior:

$$\begin{aligned} \theta_{\text{MAP}} &:= \text{argmax}_{\theta \in \Theta} \, p(\theta \mid \mathcal{D}) \\ &= \text{argmin}_{\theta \in \Theta} - \log p(\mathcal{D} \mid \theta) - \log p(\theta) \end{aligned}$$

A sharply concentrated prior helps to reduce the variance of the resulting estimator. However, if our a priori belief is mis-specified, this can increase the bias.

Theorem 2.7.1
Let $p(\theta)$ be a prior pdf of the parameter θ, $p(\mathcal{D} \mid \theta)$ the pdf of data $\mathcal{D}$ given θ, and $p(\mathcal{D}) = \int p(\theta)p(\mathcal{D} \mid \theta)d\theta$ the data pdf.
Then

$$p(\theta \mid \mathcal{D}) = \text{argmin}_{q(\theta)} \, \text{KL}(p(\mathcal{D})q(\theta) \| p(\theta)p(\mathcal{D} \mid \theta)),$$

where the minimization is over all pdfs $q(\theta)$.

This result shows that Bayes rule arose from optimization.

Part II

Neural Networks

Chapter 3

Multi-Layer Perceptron

3.1 Multi-Layer Perceptron

3.1.1 Failure of Perceptron

Recall that there is no separating hyperplane for the XOR problem

$$\{(0,0),(0,1),(1,0),(1,1)\}.$$

In order to address this, we can transform the data and keep the linear model (kernel methods), or fix the data and enrich the model. While kernel methods require us to choose a feature transformation beforehand, neural networks seek to learn the representation!

3.1.2 Two-Layer Perceptron

Given an input x, we learn weights U, c, w, b. The algorithm computes

$$\begin{aligned} z &= Ux + c \\ h &= f(z) \\ \hat{y} &= \langle h, w \rangle + b \end{aligned}$$

Here f is some scalar-valued non-linear *activation function* which is univariate but applied element-wise over all entries of z.

3.1.3 Multi-Layer Perceptron

In order to generalize the two-layer perceptron, we simply stack multiple of them together.

$$\begin{aligned} z^{(1)} &= U^{(1)}x + c^{(1)} \\ h^{(1)} &= f(z^{(1)}) \\ z^{(2)} &= U^{(2)}h^{(1)} + c^{(2)} \\ h^{(2)} &= g(z^{(2)}) \\ &\cdots \\ \hat{y} &= \langle h^{(n)}, w \rangle + b \end{aligned}$$

This is also referred to as a *feed-forward neural network*.

3.1.4 Activation Function

Some popular choices include

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

Tanh $\tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$

Rectified Linear (ReLu) $t_+ = \max(t, 0)$

The sigmoid and tanh functions are smooth and thus are differentiable. However, they saturate very easily and are susceptible to the *vanishing gradient problem*. The ReLu function does not suffer from this but it is not differentiable at 0.

3.1.5 Underfitting vs Overfitting

Linear predictors tend to underfit the data. On the other hand, neural networks learn hierarchical nonlinear features jointly with a lienar predictor. Such flexibility easily leads to overfitting. There are heuristics to address this in practice.

3.2 Automatic Differentiation

In order to train the weights, we view a prediction as

$$\hat{y} = q(x; \Theta)$$

where x is the input and Θ is the weights we must train.

We need a loss function ℓ to measure the difference between the predicted truth $\hat{y}$ and truth y. For regression we can simply use the squared loss. For classification, we can use the logistic loss function.

3.2.1 Gradient Descent

Our goal is to solve the minimization problem

$$\min_{\Theta} L(\Theta) := \frac{1}{n} \sum_{i=1}^{n} \ell[q(x_i; \Theta), y_i].$$

We can perform gradient descent iteratively with the update rule

$$\Theta_{t+1} := \Theta_t - \eta_t \boldsymbol{\nabla} L(\Theta_t).$$

3.2.2 Stochastic Gradient Descent

Computing the average of gradients over all inputs is expensive. We can instead sample a single data point randomly and apply the update rule

$$\Theta_{t+1} := \Theta_t - \eta_t \boldsymbol{\nabla} \ell[q(x_{i_t}; \Theta_t), y_{i_t}].$$

Another benefit of SGD is that the weights we trained tend to generalize better.

We can apply a diminishing step size $(1/\sqrt{t}, 1/t)$. There are also more sophisticated methods such as averaging, momentum, variance-reduction, etc. When sampling the input, we can sample without replacement, cycle, or permute in each pass.

3.2.3 Automatic Differentiation

For all the gradient optimization methods, we need to actually compute the gradient. AutoDiff (backpropagation) is an efficient way to compute the gradients in linear time based on the chain rule.

Definition 3.2.1 (Function Superposition & Computational Graph)
Let $\mathcal{F}$ be a class of basic functions. A vector-valued function $g : X \subseteq \mathbb{R}^d \to \mathbb{R}^m$ is a superposition of the basic class $\mathcal{F}$ if

(i) There is some DAG $G = (V, E)$ with topological sorted order

$$\underbrace{v_1, \ldots, v_d}_{\text{input}}, \underbrace{v_{d+1}, \ldots, v_{d+k}}_{\text{intermediate vars}}, \underbrace{v_{d+k+1}, \ldots, v_{d+k+m}}_{\text{output}},$$

and $v_i v_j \in E$ implies that $i < j$.

(ii) For each node v_i, let $I_i := \{u \in V : (u, v_i) \in E\}$ and $O_i := \{u \in V : (v_i, u) \in E\}$ denote teh predecessor and successors of v_i, respectively.

(iii) The nodes are computed sequentially for $i = 1, \ldots, d + k + m$ as follows

$$v_i = \begin{cases} x_i, & i \leq d \\ f_i(I_i), & i > d \end{cases}$$

where each $f_i \in \mathcal{F}$.

Let $\mathcal{F} := \{+, \times, \sigma, \text{constant}\}$. Then any multi-layer NN is a superposition of the basic class $\mathcal{F}$.

Theorem 3.2.1 (Automatic Differentiation)
Let $\mathcal{F}$ be a basic class of differentiable functions including $+, \times$, and all constants. Denote $T(f)$ as the complexity of computing f and $T(f, \boldsymbol{\nabla} f)$ the complexity of additional computation of its gradient.
Let I_f, O_f be the input and output arguments and assume there exists some constant $C = C(\mathcal{F}) > 0$ so that for all $f \in \mathcal{F}$,

$$T(f, \boldsymbol{\nabla} f) + |I_f| \cdot |O_f| \cdot [T(+) + T(\times) + T(\text{constant})] \leq C \cdot T(f).$$

Then for any superposition $g : \mathbb{R}^d \to \mathbb{R}^m$ of the basic class $\mathcal{F}$,

$$T(g, \boldsymbol{\nabla} g) \leq C\gamma(m \wedge d) \cdot T(g)$$

where γ is the maximal output dimension of basic functions used to superpose g.

The proof of this theorem is forward differentiation when the input size is smaller than the output size and backward differentiation when the input size is bigger than the output size.

Thus, surprisingly, for real-valued superpositions ($m = \gamma = 1$), computing the gradient ($d \times 1$-dimensional vector) takes $O(d)$ time. This is at a tradeoff for space complexity since the values at each node can be computed on the fly but we need a forward pass to collect

and store function values at each node for backward mode.

3.2.4 Some Remarks

If we assign an edge weight

$$w_{ij} = \frac{\partial v_j}{\partial v_i}$$

to each $(i, j) \in E$ in our computational graph, then the desired gradient of the superposition g is

$$\frac{\partial g_i}{\partial x_j} = \sum_{\text{path } P\,:\, v_j \to v_i} \prod_{e \in P} w_e.$$

However, we cannot compute the above naively, since the number of paths in a DAG can grow exponentially with the depth. Forward and backward differentiation correspond to two dynamic programming solutions for computing this sum of products.

It is known that finding the optimal way to compute the above is in fact NP-hard.

It is also known that the dependence on dimension $m \wedge d$ cannot be reduced in general.

The proof of Theorem 3.2.1 depends only on the chain-rule property of differentiation. We could replace differentiation with any other operation which respects the chain rule such as relative differential, directional derivative, and even the Hessian-vector multiplication.

3.3 Universal Approximator

Theorem 3.3.1
Any continuous function $g : [0, 1]^d \to \mathbb{R}$ can be uniformly approximated in arbitrary precision by a two-layer NN with an activation function that is not a polynomial.

In fact, if we allow an arbitrary number of layers, any activation that is not affine will suffice!

Many kernels are universal as well. So universality is desirable but not the main reason why NNs are so effective. Indeed, one caveat is that we may need exponentialy many hidden units to get good approximations. On the other hand, increasing the depth of the network may reduce network size exponentially.

3.4 Vanishing Gradients

What can go wrong in practice when training deep neural networks?

Consider the following activation functions

Sigmoid $\sigma(x) := \frac{1}{1+e^{-x}}$ with derivative $\sigma(x)[1-\sigma(x)]$.

Tanh $\tanh(x)$ with gradient $1-\tanh^2(x)$.

As $x \to \infty, -\infty$, both derivatives tend towards 0! Now, when we compute the gradients of the weights in a deep neural network, we are essentially considering a large product of derivatives. If any of the terms are in the product, the gradient becomes very small! If we use SGD to train our network, we get stuck at this point. This is call the *vanishing gradient problem*.

There are many ways to help combat this problem. Typically, we initialize weights close to 0. We can design our neural networks with some specific structure to avoid the vanishing gradient problem We should also attempt to choose activation functions that do not saturate.

We tend to avoid sigmoid and tanh functions in practice due to the vanishing/exploding gradients. By default, we try ReLU and explore other functions if ReLU results in dead hidden units.

3.4.1 ReLU

$$f(t) := \max(0, t).$$

Recall the ReLU function. It is computationally efficient. However, when $x < 0$, its gradient is 0 and we cannot update parameters. We can initialize ReLu units with slightly positive values, or attempt other activation functions.

3.4.2 Leaky ReLU

$$f(t) := \max(0.1t, t).$$

The leaky ReLU function is also computationally efficient and has the additional benefit of having a constant gradient for both $x > 0, x < 0$.

3.4.3 ELU

$$f(t) = \begin{cases} t, & t \geq 0 \\ \alpha(e^t - 1), & t \leq 0 \end{cases}$$

This activation function has a small gradient at very negative inputs.

3.5 Overfitting & Regularization

Due to the flexibility of neural networks, we can easily overfit to our training data. It is not uncommon that we have more weights than data points! Regularization is used to combat this.

3.5.1 L2/L1 Regularization

This option merits no further explanation.

3.5.2 Dropout

For each training example, we delete hidden units with probability p. This essentially uses a smaller (induced) subnetwork for each training example (less capacity). Alternatively, we can also delete edges directly.

Consider an implementation of dropout in a single layer. Let us denote the output of this layer as $h \in \mathbb{R}^k$

1) Generate $d \in \{0, 1\}^k$ indicating whether units are kept or not.
2) Update $h := h \odot d$
3) Update $h := h/p$ (inverted dropout)

Note that with the inverted dropout step, the value of h is unchanged in expectation.

Dropout is usually only done during training. For predictions, we tend to use the entire network as-is.

Intuitively, we prevent the network from relying on any particular unit since it can be dropped. Thus we force the weights to be spread out, such as in L2 regularization.

Another interpretation is that dropout trains a large ensemble of models which share parameters.

Some ideas for choosing p are keeping p the same for all layers, design a p specifically for each individual layer (wider layer means dropping more), or we can also design p for the particular input. Usually, we drop very infrequently.

3.5.3 Data Augmentation

The best way to improve generalization is to have more training data. In the absence of this, we can create artificial data.

We are attempting to usher the weights towards "invariance" against slight changes in data.

3.5.4 Early Stopping

For models with large capacity, training error decreases steadily but validation error decreases and then increases. We should stop the training once the validation error stops decreasing.

It is a good practice when training large models to save snapshots of the weights. Once we detect overfitting, we can backtrack to use any of the previously saved snapshots. In fact, we can even do "bagging" with each of the previously saved snapshots.

3.5.5 Batch Normalization

We can apply batch normalization in the hidden layers either before or after the activation function.

For each feature/dimension over a mini-batch, we substract the its mean and divide by its standard deviation. This ensures that the data is both centered at 0 and does not contain too many extreme values.

$$\hat{x} := \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}.$$

The ϵ is there for numerical stability.

Then, we then shift and re-scale:

$$y := \gamma \hat{x} + \beta$$

where γ, β are learned scale and shift variables.

Notice that for a fixed γ, β, the mean and standard deviation of a particular layer is fixed. More importantly, it is independent of the previous layer's parameters. In essence, this makes the current layer rebust to changes in the previous layer and induces independent learning of each layer.

An additional benefit of batch normalization is to speed up learning. If the scales of features are very different, any choice of step size can lead to slow learning or over shooting for at least one feature. Batch normalization mitigates this risk.

Finally, as an unintended effect, batch normalization has a slight regularization effect. Each mini-batch being scaled and shifted adds some noise to the input to activation functions and to each layer's activations.

3.6 Optimization

3.6.1 Momentum

SGD is cheap to perform but converges relatively slowly. We can apply a common trick called *momentum*. Suppose we have a mini-batch of size m.

1) $g := \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)}) \right)$
2) $v := \alpha v + (1 - \alpha) g$
3) $\theta := \theta - \epsilon v$

Here g is the gradient over the mini-batch and v is a weighted average over all previous stochastic gradients. The hyperparameter α decides how quickloy the contributions of previous gradients decay ($\alpha = 0.9$ tends to work well).

3.6.2 Root Mean Square Propagation (RMSProp)

The idea behind this algorithm is to have a different adaptive step size for each individual weight. Suppose we have a mini-batch of size m.

1) $g := \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)}) \right)$
2) $r := \rho r + (1 - \rho) g \odot g$
3) $\Delta\theta := -\frac{\epsilon}{\sqrt{r} + \delta} \odot g$
4) $\theta := \theta + \Delta\theta$

Here g is the gradient over the mini-batch, ϵ is some global learning rate, and ρ is the decay rate.

3.6.3 Adaptive Moments (Adam)

We can also combine both ideas. Suppose we have a mini-batch of size m.

1) $g := \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)}) \right)$
2) $s := \rho_1 s + (1 - \rho_1) s$
3) $r := \rho_2 r + (1 - \rho) 2) g \odot g$
4) $\hat{s} := \frac{\hat{s}}{1 - \rho_1^t}$ (correct bias)
5) $\hat{r} := \frac{\hat{r}}{1 - \rho_2^t}$ (correct bias)
6) $\Delta\theta := -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$
7) $\theta := \theta + \Delta\theta$

Here g is the gradient over the mini-batch, t is the iteration number, ϵ is some global learning rate, and the ρ_i's are decay rates.

Note that when $t \to \infty$, the effect of bias correction is small.

3.6.4 Learning Rate

Most popular training algorithms have the learning rate as a hyperparameter.

If we tune step sizes by hand, we typically reduce the learning rate over time every time the algorithm is stuck.

There are also adaptive step size ideas such as step decay (halving every few epochs), exponential decay ($\alpha = \alpha_0 e^{-kt}$), as well as inverse decay ($\alpha = \alpha_0 / (1 + kt)$).

Chapter 4

Convolutional Neural Networks

4.1 Convolutional Neural Networks

4.1.1 Convolution

In Fourier analysis, a convolution is a transformation which combines two signals into a single output signal. In neural networks, a convolution denotes a linear combination of a subset of units based on a specific pattern of weights.

$$z_j = \sum_i w_{ji} h_i.$$

Convolutions are typically combined with an activation function to produce a feature

$$h_j = f(z_j).$$

What is the purpose of this concept? Consider applying MLPs for images. The number of weights required are enormous, leading to overfitting. Convolutional neural networks aim to resolve this issue by exploting the strong spatial local correlation present in natural images. Instead of fully connected weights, a convolutional layer has *kernels (filters)* that are only connected to a small region of the previous layer via dot products. Each filter can be thought of as "grouping" a region from the previous region into a single neuron.

4.1.2 Filters

Imagine a $32 \times 32 \times 3$ image and a $5 \times 5 \times 3$ filter. We convolve the filter with the image by sliding the filter over the image spatially and computing dot products. This results in a $28 \times 28 \times 1$ tensor as activaation map output.

It is important to note that these filters are shared. Moreover, they are learned from data.

We can also use multiple filters. Suppose we used 6 filters instead of just 1. Then the output is a $28 \times 28 \times 6$ tensor.

After putting the output tensor through an activation function, we can then repeat the convolution. Say we now use 10 filters each with dimension $5 \times 5 \times 6$. The result is a $24 \times 24 \times 10$ tensor.

Note that the number of channels of a filter is always equal to the number of channels of the input!

4.1.3 Why Convolve?

As previously mentioned, parameter sharing across filters allow us to greatly decrease the number of weights we need to train.

One interpretation is that repeated convolutions extract higher-level features which are relevant to the ML task at hand. For image recognition, the pixel input is difficult to interpret entry-wise. However, after several convolutions, each neuron is computed from large areas of the original input, which can extract more high-level features of the image.

4.1.4 Convolution Output

Three hyperparameters control the output size.

depth is the number of filters.

stride determines how to slide the filters such as 1 or 2 pixels at a time.

zero-padding determines how much padding we add to the input and allows control of the spatial size of the output

Note that we typically choose to preserve the spatial size of the input. In the event we do choose to apply padding, there is no restriction on using zeros. We can for example make a copy of the first column.

Let W be the width of the input, F the with of the filter, P, the amount of padding across the width, and S the stride length across the width. If we pad both sides equally, the output width is

$$\left\lfloor \frac{W + 2P - F}{S} + 1 \right\rfloor.$$

The computation for the height is identical.

In practice, we set $P = (F-1)/2$ and F to be an odd integer. This ensures that the input and output have the same spatial size.

4.1.5 Pooling

Another aspect of convolutional neural networks is pooling. The idea is to perform down-sampling along the spatial dimensions. This reduces the amount of parameters and computation as well as prevents overfitting.

Here are some common operations.

max-pooling is the most popular approach (max-pooling with 2×2 filter size and stride length 2)

average pooling

$L2$**-norm** pooling

One benefit of pooling is making the output invariant to slight perturbations. Consider max pooling: unless the maximum changes in the small square, the pooled output remains the same!

One difference between convolution and pooling is that pooling is always applied separately to each channel whereas we take the dot product for convolutions across a channels (within a specific spatial region).

4.1.6 Layers in CNNs

Fully-Connected Layers

As we have seen before, each unit in these layers is connected to all units in the previous layer.

Convolutional Layer

In summary, a convolutional layer accepts a tensor of size $W_1 \times H_1 \times D_1$ and has K filters, each with spatial extend F, taking stride length S, and adds zero padding P. It produces outputs of size $W_2 \times H_2 \times D_2$ with

$$W_2 = \left\lfloor \frac{W_1 - F + 2P}{S} + 1 \right\rfloor$$

and similarly for H_2. $D_2 = K$.

With parameters sharing, it introduces $F \cdot F \cdot D_1$ weights per filter for a total of $F \cdot F \cdot D_1 \cdot K$ weights and K biases.

Pooling Layer

In summary, a pooling layer accepts a tensor of size $W_1 \times H_1 \times D_1$. It consists of a spatial extend F as well as the stride length S. The layer produces a volume of size $W_2 \times H_2 \times D_2$ where

$$W_2 = \left\lfloor \frac{W_1 - F}{S} + 1 \right\rfloor$$

and similarly for H_2. We always have $D_2 = D_1$.

Note that a pooling layer introduces no parameters and that it is uncommon to use zero-padding for pooling layers.

Typically, we set the stride length equal to the filter size so each pixel of the output corresponds to a distinct region of the input.

4.2 Network Structure

A typical convolutional neural network has the following structure:

1) Input
2) (Repeat)
    (a) (Repeat) Convolution to ReLU
    (b) Pooling
3) Fully Connected Layer to ReLU
4) Fully Connected Layer

4.2.1 LeNet

This network was invented by LeCun et al. in 1998 to detect handwritten characters.

The architecture is CONV-POOL-CONV-POOL-FC-FC. Convolutional filters were 5×5, applied at stride 1. Subsampling (pooling) layers were 2×2 applied at stride 2.

4.2.2 AlexNet

This structure was due to Krizhevsky et al. in 2012 and sparked the recent interest in deep learning.

The architecture is CONV1-POOL1-NORM1-CONV2-POOL2-NORM2-CONV3-CONV4-CONV5-POOL3-FC6-FC7-FC8. It features the first use of the ReLU function in practice. AlexNet also used norm layers which are not common anymore. Moreover, it employed heavy data augmentation, dropout, batch training with SGD momentum

This network was also one of the first to use GPUs for training. At the time, GPU memory was limited so there was significant engineering efforts to split the network among two GPUs.

4.2.3 VGGNet

While the general structural differences between LeNet and AlexNet were minimal, this network by Simonyan and Zissermaan in 2014 changed the perspective of convolutional neural networks.

VGGNet advocated for smaller filters and deeper networks. There are also significantly more convolutions and as a result many more layers.

The observation here is that using 3×3 filters three times is in some sense equivalent to using a 7×7 once. However, the number of parameters is $3^2 \times d \times 3$ where d is the number of input channels, whereas the 7×7 filter uses $7^2 \times d$ parameters which is strictly more!

There are three factors to consider when designing neural networks: number of parameters, memory consumption, and necessary computatation. An interesting observation for VGGNet is that as we follow the input through the network, the memory per layer decreases while the number of parameters increases. This is due to the intermediate "images" shrinking as a result of convolutions, while the number of parameters slowly increases from the chosen number of filters and significantly increasing once we reach the fully connected layers.

4.2.4 GoogLeNet

The authors Szegedy et al. wanted to create even deeper models. The naive *inception module* involves using multiple choices of filter sizes for a convolutional layer and concatenating them together at the end. The idea behind this is that at least one of the choices will lead to a good representation.

The actual model adds some 1×1 "bottleneck" layers before applying filters and after pooling for dimension reduction. This can reduce the computation required.

The full GoogLeNet stacks these inception modules upon each other to form a very deep

network. There is also no fully connected layers due to the large number of parameters. Instead, the last layer consists of a simple pooling operation where each channel becomes an entry of the output vector.

4.2.5 ResNet

In theory, deeper networks should result in better performance, at least on the training set. However, deeper models are harder to optimize in practice (vanishing/exploding gradients) and in fact leads to worse performance in reality.

A solution is to add some "identity shortcut conection" that skips one or more layers. More precisely, we add the output of a previous hidden layer (say $\ell - 3$) to the output of a convolutional layer (say the current layer ℓ) before applying the activation function.

Chapter 5

Recurrent Neural Networks

5.1 Recurrent Neural Networks

5.1.1 Motivation

Convolutional neural networks can be applied to variable size inputs! As long as we only have convolutions and pooling in our network, we are able to avoid the issue of fixed dimensions which come with multilayer perceptrons. We can also train convolutional neural networks much more efficiently since we can leverage GPU parallelization.

Consider an image captioning task, where we are given images and the goal is to automatically generate a textual description of the image. Here the input is of fixed dimension, but the output is not.

On the other hand, consider a sentiment classification problem, where we are given textual input and the goal is to determine whether the sentence is positive or negative. The input here is variable but the output dimension is fixed.

Finally, an example of a variable input and variable output problem is machine translation. The goal is to produce a translation between two languages given a phrase in one language. Another such is frame level image classification where we need to provide an output immediately when going through a video and possibly updating this output as the video is being played.

5.1.2 Parameter Sharing

Recurrent neural networks heavily rely on parameter sharing. We can think of an output as

$$\begin{aligned} h^{(t)} &= g^{(t)}(x^{(t)}, x^{(t-1)}, \dots, x^{(1)}) \\ &= f(h^{(t-1)}, x^{(t)}, \theta). \end{aligned}$$

Here $h^{(t-1)}$ is the *memory*, or old state at time $t-1$. $x^{(t)}$ is the new input at time t. Finally, θ is the set of parameters which are shared at every step. The function f is used at every step!

Example 5.1.1
At each time step, the RNN produces an output and has recurrent connections between hidden units.

$$\begin{aligned} a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)} \\ h^{(t)} &= \tanh\big(a^{(t)}\big) \\ o^{(t)} &= c + Vh^{(t)} \\ \hat{y}^{(t)} &= \text{softmax}(o^{(t)}) \end{aligned}$$

At each step $\hat{y}^{(t)}$ is the output.

5.1.3 Training & Testing

Training is done through computing the gradients via back-propagation. At test-time, sample characters one at a time, and feed it back into the model. Thus $x^{(t)} = \hat{y}^{(t-1)}$.

5.2 Sequential vs Parallel

Sequential RNNs are very powerful. In fact, they are Turing complete! However, they are cumbersome to train. We can instead change the recurrent

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, \theta) \mapsto h^{(t)} = f(o^{(t-1)}, x^{(t)}, \theta)$$

This is not yet parallelizable. However, we can employ another change during training:

$$h^{(t)} = f(\hat{y}^{(t-1)}, x^{(t)}, \theta).$$

Intuitively, this makes sense since a good output o should be similar to the prediction y. At test-time, we still use the output o.

5.3 Vanilla RNN Gradient Problems

Recall the vanishing and exploding gradient issues from MLP. This issue is even more prevalent in RNNs.

5.3.1 Clipping

One simple solution is clipping: If the gradient g is too big, say $\|g\| > v$, set $g := \frac{gv}{\|g\|}$. Here v is some threshold.

Similarly, the same can be done for gradients which are too small.

5.3.2 Long Short Term Memory (LSTM)

Another common solution is to create paths along which the product of gradients is near 1. Thus we resort to fancier architecture.

Say our recurrence resembles the following

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W \begin{bmatrix} h^{t-1} \\ x^{(t)} \end{bmatrix}.$$

Here we have

- i: Input gate, whether to write to cell
- f: Forget gate whether to keep previous raw memory
- g: Gate gate, new raw memory
- o: Output gate, How much to reveal cell

Then we set

$$\begin{aligned} c_t &= f \odot c_{t-1} + i \odot g \\ h^{(t)} &= o \odot \tanh(c_t) \end{aligned}$$

Now, backpropagation from c_t to c_{t-1} involves only element-wise multiplication and addition.

5.3.3 Gated Recurrent Unit (GRU)

GRU can be thought of as a simplication of LSTMs. The important detail is to take a convex combination of previous and new raw memory.

$$
\begin{aligned}
r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\
z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\
\tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}
$$

5.4 Other Architectures

5.4.1 Bidirectional RNNs

It may make sense to allow information to flow from present to past. We will have

$$
\begin{aligned}
h^{(t)} &= f_h(h^{(t-1)}, x^{(t)}, \theta) && \text{past} \\
g^{(t)} &= f_g(h^{(t+1)}, x^{(t)}, \theta) && \text{future} \\
o^{(t)} &= f_o(h^{(t),g^{(t)}}, \theta) && \text{output}
\end{aligned}
$$

5.4.2 Deep RNNs

We might consider taking hidden states from time steps further than just $t - 1$. Usually, there is 2-3 hidden layers.

5.4.3 Encoder-Decoder

This architecture is widely used for neural machine translation.

We have an encoder RNN which maps variable dimension input to a fixed dimension context. Then, another decoder RNN maps the context to a variable dimension output.

Chapter 6

Graph Neural Networks

6.1 Graph Neural Networks

Let $G = (V, E)$ be a graph and l some features. We can compute a state (embedding) of G as

$$x_n = f_w(x_{N(n)}, l_{N(n)}, l_{\delta(n)}).$$

Then, we output $o_n = g_w(x_n, l_n)$.

Here we can think of f, g as neural networks with parameter w which is shared among nodes.

Note that this is a recurrent system of equations which we can write as

$$x = F_w(x, l).$$

If F is a contraction, then there exists a unique state x. We can compute this solution iterative with the update rule

$$x^{(t+1)} = F_w(x^{(t)}, l).$$

Upon convergence, we take a network G_w and output $G_w(x, l)$.

6.2 Graph Convolution Networks

Recall CNNs perform convolutions on grid graphs. How can we convolve in general graphs?

6.2.1 Spatial Convolution

One simple way to go about this is to order the neighbors at each node and choose a fixed number of them to compute the convolution.

6.2.2 Spectral Convolution

Recall the graph Laplacian L is a real symmetric matrix. Thus we can decompose

$$L = U\Lambda U^T.$$

Given two graph signals $x, g \in \mathbb{R}^{|V|}$, we can compute

$$x * g := U[(U^T x) \odot (U^T g)].$$

More generally, we define one layer of spectral convolution as

$$x_r^{(l+1)} = \sigma(U[W_r^l \odot (U^T X^l)]\mathbb{1}).$$

Here $r = 1, \ldots, d_{l+1}$ where $X^l = [x_1^l, \ldots, x_{d_l}^l]$. The filter weights W_r^l are learned.

Chebyshev Net

spectral convolution requires eigen-decomposition which is an expensive non-local operation. We can rewrite the convolution

$$\begin{aligned} x * g &= U[(U^T g) \odot (U^T x)] \\ &= U[\operatorname{diag}(f(\lambda; w))(U^T x)] \\ &= [U \operatorname{diag}(f(\lambda; w))U^T]x. \end{aligned}$$

Thus we view the transform as a function of the eigenvalues. In particular, choosing f to be a polynonomial dispenses eigen-decomposition.

Suppose $f(\lambda; w) = \sum_{j=0}^k w_j \lambda^j$. Then the operation becomes taking powers of the Laplacian

$$\sum_j w_j L^j x.$$

This is a local operation requiring only k hops!

6.2.3 Graph Convolutional Nets

A layer of GCN is simply

$$X^{l+1} = \sigma(D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X^l W^l)$$

where $X^l \in \mathbb{R}^{|V| \times s}$ is the stack of features at layer l and W^l are the weights. Note that A, D are the adjacency and degree matrices after adding self-loops.

Now, one layer of GCN only aggregates information from 1-hop neighbors. We need to stack to get information from k-hop neigbors.

Weifeiler-Lehman Algorithm

We can think of GCNs as a special case of the Chebyshev net. Another way is to observe that it is a special case of GNNs. Moreover, it resembles the color refinment algorithm by Weisfeiler and Lehman.

6.3 Graph Generative Models

Give a set of graphs G_i sampled from some unknown distribution, we wish to learn a model that can generate similar graphs.

6.3.1 GraphRNN

The idea is to generate the adjacency matrix sequentially. When generating the i-th row, we use an RNN

$$h_i = f_w(h_{i-1}, A_{i-1}).$$

In particular, we repeated ask ourselves "should the new node be connected with some previous node?"

6.3.2 Graph Normalization Flow

We first train a standard encoder decoder for our graph dataset. At test time, we push a standard normal distribution through the GNF, generating some hidden state. Then, this hidden state is fed to a decoder.

6.4 Graph Robustness

When we perturb the graphs, how does the model change?

Part III

Miscellaneous Models

Chapter 7

k-Nearest Neighbors

7.1 Algorithm

7.1.1 1-NN

At training time, we simply store the training set. This takes $O(1)$ auxiliary time.

At test time for query x, we query the closest point in the training set and output its corresponding y. This incurs a running time of $O(nd)$.

7.1.2 Voronoi Diagram

There is a huge imbalance of running time between training and testing. In $\mathbb{R}^d$, we can construct a *voronoi diagram* which uses $n^{O(d)}$ space and supports queries in $O(n \log n)$ time. This reduces the testing time but still suffers from the curse of dimensionality.

7.1.3 k-NN

The general version is similar. We again store the entire training set at training time. At test time, we query the k closest points and aggregate their labels. For classification, this could be a majority vote, for regression, this could be some average.

The hyperparameter k controls the bias-variance tradeoff. when $k = 1$, there is high variance. However, when $k = n$, we always predict the average of the dataset, which has high bias.

7.2 Theory

7.2.1 Baye's Rule

Recall the Baye's error

$$P^* := \min_{f:X\to\{\pm 1\}} P(f(X) \neq Y)$$

which is the minimum classification error.

The Baye's classifier is given by the posterior probability

$$\eta(X) = P(Y = 1 \mid X)$$

and we predict $f^*(X) = 1$ if and only if $\eta(X) \geq \frac{1}{2}$.

This generalizes to c-class classificiation where

$$P^* = \mathbb{E}[1 - \max_{m\in[c]} P(Y = m \mid X)]$$

and the prediction is given by

$$f^*(X) = \text{argmax}_{m\in[c]}\, P(Y = m \mid X).$$

This is the absolute best we can do even when we know the distribution of (X, Y).

Theorem 7.2.1 (Cover, Hart; '67)
Let $Y_k^{(n)}$ denote the k nearest neighbor predictor on a dataset of size n. Then

$$\lim_n P(Y_1^{(n)} \neq Y) \leq 2P^* - \frac{c}{c-1}(P^*)^2.$$

Note that this only holds in the limit!

7.2.2 1-NN vs k-NN

Suppose now that our data is generated uniformly randomly from $B_{\frac{1}{4}}(-1, 0) \cup B_{\frac{1}{4}}(1, 0)$ and the label is the sign of the first coordinate.

The error of the 1-NN is simply $\frac{1}{2^n}$ since it only makes a mistake if all training points came from the opposite ball to the test point. On the other hand, for $k = 2t + 1$, the k-NN error becomes $\frac{1}{2^n}\sum_{i=0}^{t}\binom{n}{i}$, which is strictly more than 1-NN.

Empirically, this suggests that well separated data sets should use lower k while complicated datasets with overlapping classes should use higher k.

7.2.3 Curse of Dimensionality

Theorem 7.2.2
For any $c > 1$ and learning algorithm L, there exists a distribution over $[0,1]^d \times 0, 1$ such that the Bayes error is 0 but for sample size $n \leq \frac{1}{2}(c+1)^d$, the error of the rule L is greater than $\frac{1}{4}$.

Thus k-NN is effective when there are MANY training samples. Dimensionality reduction may prove to be helpful.

7.3 Applications

7.3.1 Locally Linear Embedding

Let $N(x_i)$ denote the k nearest neighbors of the training point x_i. First, we solve the minimization problem

$$\min_{W\mathbb{1}=\mathbb{1}} \left\| x_i - \sum_{x_j \in N(x_i)} W_{ij} x_j \right\|_2^2.$$

Then, we solve for a smaller dimension embedding

$$\min_z \sum_i \left\| z_i - \sum_j W_{ij} z_j \right\|_2^2.$$

Chapter 8

Boosting & Bagging

8.1 Choosing Algorithms

How can we choose the best ML algorithm for a particular application?

We can simply try them all and choose the "best" one, but how about combining existing algorithms?

8.2 Boostrap Aggregating (Bagging)

Given a dataset $\mathcal{D} = \{(x_i, y_i) : i \in [n]\}$, we can generate T datasets, each of size n, by sampling from $\mathcal{D}$ with replacement. Then, we can fit T different classifiers and combine them through majority voting.

In principle, we can apply bagging for any algorithm. However, simple algorithms are used in practice. This is because we need to train T different models and expensive algorithms such as deep neural networks simply will not scale.

The intuition behind bagging is that IF each bootstrapped dataset was independent, taking an average or majority voting reduces the variance. Even though we do not have independence in bagging, the hope is that variance will still be somewhat reduced.

8.2.1 When Does Bagging Work?

Bagging is essentially averaging. This is especially beneficial when base classifiers have high variance (decision trees). On the other hands, algorithms which have low variance and high bias would not benefit (k-NNs).

8.3 Randomizing Output

Given a dataset $\mathcal{D} = \{(x_i, y_i) : i \in [n]\}$, we can generate T datasets, by adding small Gaussian noise to y. Then, we can fit T different regressors and average their outputs.

For classification, we can encode the labels and reduce to regression. Alternatively, we can randomly change a small proportion of labels in the training set.

This intuition behind randomizing output is that it makes it more difficult for each individual agorithm to overfit.

8.4 Random Forests

Recall the decision algorithm, where at each internal node, the "best" feature is chosen to split the data.

Random forests generalize by adding two levels of randomness. First, we apply bagging to train T different trees over which we take the majority vote. Secondly, instead of considering all features at each internal node, we consider a random subset of features instead.

8.5 Boosting

8.5.1 Hedging

We can think of this as the online experts problem. We have N "experts" and our goal is bet on the best expert on the fly.

Choose a parameter $\beta \in (0, 1)$. Initialize weights $w^{(1)} \in [0, 1]^N$. For $t = 1, \ldots, T$:

1) Let $p^{(t)} := 2^{(t)} / \sum_{i=1}^{n} w_i^{(t)}$
2) Receive loss from environment $\ell^{(t)} \in [0, 1]^N$
3) Suffer loss $p^{(t)} \cdot \ell^{(t)}$
4) Update weights $w_i^{(t+1)} := w_i^{(t)} \beta^{\ell_i^{(t)}}$

Proposition 8.5.1
We have

$$\sum_{t=1}^{T} p^{(t)} \cdot \ell^{(t)} \leq \frac{\ln N - \ln \beta \cdot \min_{i \in [N]} \sum_{i=1}^{T} \ell_i^{(t)}}{1 - \beta}$$

Corollary 8.5.1.1
By choosing an appropriate β, we have

$$\begin{aligned}\frac{1}{T}\sum_{i=1}^{T} p^{(t)} \cdot \ell^{(t)} &\leq \min_{i\in[N]} \frac{1}{T}\sum_{t=1}^{T} \ell_i^{(t)} + \sqrt{\frac{2\ln N}{T} \cdot \min_{i\in[N]} \frac{1}{T}\sum_{t=1}^{T} \ell_i^{(t)} + \frac{\ln N}{T}} \\ &= \min_{i\in[N]} \frac{1}{T}\sum_{t=1}^{T} \ell_i^{(t)} + o(1).\end{aligned}$$

Note that there are no assumptions on $\ell^{(t)}$. In fact, even adversarial choices of $\ell^{(t)}$ will abide by this result. This is because adversarial choices of $\ell^{(t)}$ will increase the loss of the baseline expert.

8.5.2 Adaptive Boosting (AdaBoost)

We can think of AdaBoost as choosing the parameter β adaptively.

Suppose we have a dataset $\mathcal{D} = \{(x_i, y_i) : i \in [N]\}$ as well as a weak classifier algorithm W which accepts weighted training sets.

Initialize the weights $w_i^{(1)} \in [0,1]$ for $i = 1, \ldots, N$. Then for $t = 1, \ldots, T$:

1) Let $p^{(t)} := w^{(t)}/\mathbb{1}^T w^{(t)}$
2) Call W with weights $p^{(t)}$ and receive a hypothesis $h_t : X \to [0,1]$
3) Calculate the error of h_t given by $\epsilon_t = p \cdot |h_t(x) - y|$
4) Set $\beta_t := \epsilon_t/(1-\epsilon_t)$
5) Set the new weights vector to be $w_i^{(t+1)} := w_i^{(t)} \beta_t^{1-|h_t(x_i)-y_i|}$

Finally, we aggregate all the trained classifiers

$$h_f(x) := \begin{cases} 1, & \sum_{t=1}^{T} (\log \frac{1}{\beta_t}) \cdot h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \log \frac{1}{\beta_t} \\ 0, & \text{else} \end{cases}$$

Proposition 8.5.2
Suppose each weak classifier has error $\epsilon_t \leq \frac{1}{2} - \gamma$ for some $\gamma > 0$. Then

$$\begin{aligned}P_{i\sim\mathcal{D}}[h_f(x_i) \neq y_i] &\leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)} \\ &\leq \exp\bigl(-2T\gamma^2\bigr).\end{aligned}$$

Thus we can tink of AdaBoost as a form of gradient descent to minimize the exponential loss $\sum_i e^{-y_i h(x_i)}$ for h in the conic hull of h_t's.

With sufficiently large T, Adaboost will be able to overfit. Thus it is important to use weaker base classifiers to combat this.

8.5.3 Extensions

We can change the objective from an exponential loss to logit, l2 losses to obtain LogitBoost and L2Boost. We can also adapt AdaBoost to different problems such as multi-class classification, regression, and ranking. There is also the ever popular GradBoost.

Part IV

Generative Models

Chapter 9

Mixture Models

The goal is to learn the underlying distribution given some samples.

9.1 Mixture Models

A mixture model is parametrized by a density function.

$$p(x \mid \theta) = \sum_{k=1}^{K} p(z = k)p(x \mid z = k, \theta).$$

Here $p(z = k)$ is the *mixing distribution* given by $p(z = k) = \pi_k$ where $\pi \geq 0$ and $\sum_k \pi_k = 1$. $p(x \mid z = k, \theta)$ is the k-th *component distribution*, denoted $p_k(x \mid \theta)$.

Note that we observe x but never z.

9.1.1 Gaussian Mixture Models

Recall the multivariate Gaussian distribution with density

$$p(x) = (2\pi)^{-\frac{d}{2}}|S|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(x-\mu)^T S^{-1}(x-\mu)\right].$$

A Gaussian mixture model is parametrized as

$$p(x \mid \theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, S_k).$$

Here K is the number of components and μ, S are the parameters of the component distributions.

9.1.2 Univerality

Theorem 9.1.1
GMM with sufficiently many components can approximate any probability density function on $\mathbb{R}^d$.

Although a good theoretical result, there is no guarantee on the necessary number of components. Another remark is that the theorem does not depend on the choice of Gaussian components. However, there are computational benefits of the Gaussian distribution.

9.1.3 Identifiability

Before we raise the question of parameter estimation, is the factorization for GMMs even unique? Indeed, uniqueness holds for Gaussian components!

9.1.4 Inference

Given IID samples, our goal is to estimate the parameter θ, consisting of π_k, μ_k, S_k.

Recall that maximum likelihood distribution is equivalent to minimizing the KL divergence

$$\min_\theta \mathrm{KL}(\hat{x} \| p(x \mid \theta)).$$

Unfortunately, maximum likelihood here is NP-hard.

9.2 Expectation Maximization

Suppose we wish to minimize the log-likelihood function

$$\begin{aligned}\min_\theta \ell(\theta) &:= \sum_{i=1}^n -\log p(x_i \mid \theta) \\ &\approx \mathrm{KL}(q(x) \| p(x \mid \theta)).\end{aligned}$$

As mentioned, this is NP-hard. Perhaps unintuitively, we will introduce another variable and solve an alternating minimization problem:

$$\min_\theta \min_{q(z|x)} \mathrm{KL}(q(x)q(z \mid x) \| p(x, z \mid \theta)).$$

In other words, we match the hypothetical joint instead of marginal. Note that if we can match $p(x, z \mid \theta) = q(x, z)$, we can immediately match $p(x \mid \theta), q(x)$ by integration.

Recall the definition of the KL divergence

$$\mathrm{KL}(q(x)\|p(x)) = \int q(x) \log \frac{q(x)}{p(x)} dx \geq 0.$$

Equality holds if and only if $p = q$. It is a fact that

$$\mathrm{KL}(q(x, z)\|p(x, z)) \geq \mathrm{KL}(q(x)\|p(x)) + \mathbb{E}_x[\mathrm{KL}(q(z \mid x)\|p(z \mid x))].$$

Thus we are attempting to minimize an upperbound.

9.2.1 The Algorithm

The explicit KL minimization we are attempting is

$$\min_{q_i(z_i)} \min_{\theta} \sum_{i=1}^{n} \left[\sum_{z_i} q_i(z_i) \log q_i(z_i) - \sum_{z_i} q_i(z_i) \log p(x_i, z_i \mid \theta) \right].$$

If we fix q and solve for θ, the problem becomes

$$\min_{\theta} - \sum_{i=1}^{n} \sum_{z_i} q_i(z_i) \log p(x_i, z_i \mid \theta).$$

There is often a closed-form solution for this problem.

On the other hand, if we fix θ and solve for q:

$$\min_{q_i(z_i) \geq 0, \sum_{z_i} q_i(z_i) = 1} - \sum_{z_i} q_i(z_i) \log p(x_i, z_i \mid \theta) + \sum_{z_i} q_i(z_i) \log q_i(z_i).$$

But this is simply the KL divergence of $q_i(z_i)$ and $p(z_i \mid x_i, \theta)$! Hence the closed form solution is

$$q_i(z_i) = p(z_i \mid x_i, \theta).$$

9.2.2 EM for GMM

We write $r_{ik} := q(z_i = k \mid x_i)$. Plugging in the formula for d-dimensional Gaussian, the minimization problem becomes

$$\min_{r_{ik} \geq 0, \sum_k r_{ik} = 1} \min_{\theta} \sum_{i=1}^{n} \left[\sum_{k=1}^{K} r_{ik} \log r_{ik} - \sum_{k=1}^{K} r_{ik} \log p(x_i, z_i \mid \theta) \right].$$

Fixing q and solving for θ yields

$$\min_{\theta} \sum_{i=1}^{n} \sum_{k=1}^{K} r_{ik} \left[-\log \pi_k + \frac{1}{2} \log |S_k| + \frac{1}{2} (x_i - \mu_k)^T S_k^{-1} (x_i - \mu_k) \right].$$

By taking the derivative and setting it to 0, we get the following updates:

$$\begin{aligned}
\pi_k &= \frac{\sum_i r_{ik}}{n} \\
\mu_k &= \frac{\sum_{i=1}^{n} r_{ik} x_i}{\sum_{i=1}^{n} r_{ik}} \\
S_k &= \frac{\sum_{i=1}^{n} r_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{n} r_{ik}} \\
&= \frac{\sum_{i=1}^{n} r_{ik} x_i x_i^T}{\sum_{i=1}^{n} r_{ik}} - \mu_k \mu_k^T.
\end{aligned}$$

Finally, fixing θ and optimizing for q, we have the closed form

$$\min_{r_i k \geq 0, \sum_k r_{ik} = 1} \min_{\theta} \sum_{i=1}^{n} \left[\sum_{k=1}^{K} r_{ik} \log r_{ik} - \sum_{k=1}^{K} r_{ik} \log p(x_i, z_i \mid \theta) \right].$$

As mentioned before, this is simply the KL-divergence hence we have the closed form solution

$$r_{ik} = \frac{\pi_k |S_k|^{-\frac{1}{2}} \exp\left[-\frac{1}{2} (x_i - \mu_k)^T S_k^{-1} (x_i - \mu_k) \right]}{\sum_{c=1}^{K} \pi_c |S_c|^{-\frac{1}{2}} \exp\left[-\frac{1}{2} (x_i - \mu_c)^T S_c^{-1} (x_i - \mu_c) \right]}.$$

9.3 Applications

9.3.1 Soft Clustering

Once we compute the posterior distribution, we can use the mixing distribution as an estimate for the likelihood of belonging to a certain cluster.

9.3.2 t-Distribution

We can express the Student t-distribution as a Gaussian scale-mixture.

9.3.3 Missing Data

Since GMM is a generative model, we can impute missing data.

9.4 Mixture Density Network

Instead of using the EM algorithm, we can instead use a neural network to estimate the model.

Suppose we wish to estimate the conditional density

$$p(y \mid x) = \sum_{j=1}^{r} \lambda_j(x) p_j(y \mid x).$$

Here $p_j(y \mid x)\mathcal{N}(y \mid \mu_j(x), \sigma_j(x))$. We have a neural network which outputs the parameters

$$\begin{aligned} \lambda(x) &= \text{softmax}(g_1(x; w)) \\ \mu(x) &= g_2(x; w) \\ \sigma(x) &= \exp(g_3(x; w)). \end{aligned}$$

Then we can use maximum likelihood to estimate the weights w.

9.5 Mixture of Experts

Suppose we wish to perform regression but the data is clustered in a way where fitting multiple linear models is appropriate but a single linear model does not perform well. We can instead use mixture of experts to fit a mixture of linear models to achieve non-linearity

$$p(y \mid x, \theta) = \sum_{k=1}^{K} p(z = k \mid x, \theta) p(y \mid x, z = k, \theta).$$

Chapter 10

Generative Adversarial Networks

The goal is to learn the underlying density function given some data samples. This way, we are able to sample from the underlying distribution!

10.1 Intuition

We would like to minimize the KL divergence between population density p, and the model denqity q_θ. Assume now that we can only sample from q_θ and we cannot explicitly parametrize it. Then we cannot perform maximum likelihood estimation!

Instead, let $S : \mathbb{R}^d \to \mathbb{R}$ be some *discriminator function*. We attempt to minimize the expected discriminator score of both distributions

$$\max_S \int S(x) \cdot p(x)dx - \int f^*(S(x)) \cdot q_\theta(x)dx.$$

Since we are only able to sample points, let us replace this with the imperical average

$$\min_T \max_S \frac{1}{n} \sum_{i=1}^{n} S(x_i) - \frac{1}{m} \sum_{j=1}^{m} f^*(S(T(z_j))).$$

10.2 Push-Forward

Let $q(z)$ be some source density from which it is easy to sample, say a standard normal. Let $p(x) := (T_\# q)(x)$ be some target density we attempt to achieve. If the dimensions match, we can in fact directly compute

$$p(x) = q(T^{-1}(x)) \cdot |\boldsymbol{\nabla} T(T^{-1}(x))|^{-1}.$$

If we have T, sampling is easy: sample z from the source q and then output $T(z)$.

If the dimensions do not match, we do not have a closed-form in general. Instead, we can parameterize T as a neural network.

10.3 Fenchel Conjugate

Recall that a twice differentiable real-valued univariate function f is strictly convex if $f'' \geq 0$. The *Fenchel conjugate* of f is

$$f^*(t) = \max_s st - f(s).$$

This is always convex.

Theorem 10.3.1
f is convex if and only if $f^{**} = (f^*)^* = f$.

10.4 f-Divergence

Let f be strictly convex with $f(1) = 0$. Recall the f divergence between distributions p, q is given by

$$D_f(p\|q) := \int f\left(\frac{p(x)}{q(x)}\right) \cdot q(x)dx.$$

By the integral version of Jensen's inequality,

$$\begin{aligned} D_f(p\|q) &\geq f\left(\int \frac{p(x)}{q(x)} \cdot q(x)dx\right) \\ &= f(1) \\ &= 0. \end{aligned}$$

Here equality holds if and only if $p = q$.

Now, we rewrite f as the conjugate of the conjugate to see that

$$\begin{aligned} D_f(p\|q) &= \int \max_{S(x)} \left[S(x)\frac{p(x)}{q(x)} - f^*(S(x))\right] \cdot q(x)dx \\ &= \max_{S:\mathbb{R}^d \to \mathbb{R}} \int S(x) \cdot p(x)dx - \int f^*(S(x)) \cdot q(x)dx. \end{aligned}$$

Examples of f could be the familiar KL-divergence $f(s) = s \log s$ with conjugate $f^*(t) = \exp(t-1)$.

Another example is the Jensen-Shannon divergence $f(s) = s\log s - (s+1)\log(s+1) + \log 4$ with conjugate $f^*(t) = -\log(1-\exp(t)) - \log 4$.

10.5 Generative Adversarial Networks

All together we now have an optimization problem

$$\min_\theta D_f(p(x)\|q_\theta(x)) = \min_\theta \max_{S:\mathbb{R}^d\to\mathbb{R}} \int S(x)\cdot p(x)dx - \int f^*(S(x))\cdot q_\theta(x)dx.$$

The empirical analogue is

$$\underbrace{\min_{T_\theta}}_{\text{generator}} \underbrace{\max_{S_\psi}}_{\text{discriminator}} \frac{1}{n}\sum_{i=1}^n S_\psi(\underbrace{x_i}_{\text{true sample}}) - \frac{1}{m}\sum_{j=1}^m f^*(S_\psi(\underbrace{T_\theta(z_j)}_{\text{fake sample}})).$$

Usually, T_θ, S_ψ are both parameterized as neural networks.

10.5.1 Jensen-Shannon GAN

The original GAN used the the Jensen-Shannon divergence. With some tricks, the general optimization problem becomes

$$\min_{T_\theta}\max_{S_\psi} \frac{1}{n}\sum_{i=1}^n \log S_\psi(x_i) + \frac{1}{m}\sum_{j=1}^m \log(1 - S_\psi(T_\theta(z_j))).$$

Here $0 \le S_\psi \le 1$.

Intuitively, the generator tries to "fool" the discriminator and the discriminator in turn ppeprforms nonlinear logistic regression.

10.6 More GANs through IPM

Recall the optimizaition problem

$$\underbrace{\min_{T_\theta}}_{\text{generator}} \underbrace{\max_{S_\psi}}_{\text{discriminator}} \frac{1}{n}\sum_{i=1}^n S_\psi(\underbrace{x_i}_{\text{true sample}}) - \frac{1}{m}\sum_{j=1}^m f^*(S_\psi(\underbrace{T_\theta(z_j)}_{\text{fake sample}})).$$

If S is the class of Lipschitz continuous functions, we recover *Wasserstein GANs*. If S is the unit ball of some RKHS, we recover *MMD-GANs*. If S is the class of indicator functions, we recover *Total Variational GANs*. If S is the unit ball of some Sobolev space, we recover *Sobolev-GANs*. If S is the class of differential functions, we recover *Stein GANs*.

Chapter 11

Triangular Flows

The goal is identical to GANs where we attempt to learn a density function. However, we now assume our source and target distributions are of the same dimension.

11.1 Overview

Our goal is to obtain a diffeomorphism T between the source density q and target densitiy p.

Let $T_{\#}q$ denote the density of the warped distribution. Then it is given by

$$(T_{\#}q)(x) = q(T^{-1}(x)) \cdot \left|T'(T^{-1}(x))\right|.$$

We wish to apply maximum likelihood estimation

$$\max_T \mathscr{L}(T) := \prod_{i=1}^{n} (T_{\#}q)(x_i).$$

Thus computation of the inverse and Jacobian must both be cheap.

11.2 Increasing Triangular Maps

Recall the $\chi^2 = Z^2$ distribution is obtained by taking the square of a normal distribution. We can sample from the χ^2 distribution by simply squaring a sample from the normal distribution. Alternatively, we can warp the normal distribution to the uniform distribution using its distribution function. Then, we pass this uniform sample through the quantile

function (inverse distribution function) of χ^2. The final result is a sample from the χ^2 distribution. We will see a similar result for random vectors.

We say $T : \mathbb{R}^d \to \mathbb{R}^d$ is *triangular* if T_j is a function of only $z_1, \ldots, z_j$. The Jacobian of T is a lower-triangular matrix. This makes the determinant computation of the Jacobian very straightforward.

We say T is *increasing* if $\frac{\partial T_j}{\partial z_j}(x) > 0$ for all $j \in [d]$ and for all $x \in \mathbb{R}^d$. This makes the inverse computation relatively straightforward. Indeed, suppose $x_j = T_j(z_1, \ldots, z_j)$ for all $j \in [d]$. We first compute $z_1 = T_1^{-1}(x_1)$ using binary search. Then, we compute $z_2 = T_2^{-1}(z_1, z_2)$ using binary search with the first argument fixed. So on and so forth.

As it turns out, there is always an increasing triangular map (unique up to permutation of coordinates) which maps a source density to a target density given they are both d-dimensional.

In the 1-dimensional case, the composition of target quantile function and source distribution function is the unique T. The *Knoth-Rosenblatt transformation* generalizes this idea to an interative formula in higher dimensions.

11.3 Maximum Likelihood Estimation

Recall minimizing the KL divergence yields the maximum likelihood estimation problem

$$\min_T \sum_{i=1}^n \left[-\log q(T^{-1}(x_i)) + \sum_j \log \partial_j T_j(T^{-1}(x_i)) \right].$$

11.4 Autoregressive Models with Gaussian Conditionals

How can we paramterized these triangular maps? We can factorize the target density

$$p(x) = \prod_{j=1}^d p_j(x_j \mid x_{<j}).$$

Thus choosing a conditional implicitly fixes a family of triangular maps

$$x_j = T_j(z_j; \theta_j(z_{<j})).$$

Let us assume that

$$p(x_j \mid x_{<j}) = N(x_j; \mu_j(x_{<j}), \sigma_j(x_{<j})).$$

Then suppose $z_j \sim N(0,1)$ is a standard normal variable. We now have

$$x_j = \sigma_j(z_{<j}) \cdot z_j + \mu_j(z_{<j}) := T_j(z_j; z_{<j}).$$

Thus T_j is affine in $z_{\leq j}$.

Note that σ_j, μ_j technically depends on $x_{<j}$. However, since these depend on $z_{<j}$, the expression above is justified.

Now, the Gaussian assumption restricts us to affine triangular increasing maps. For greater expressiveness, we can allow ourselves to use GMMs.

11.5 Masked Autoregressive Flows (MAFs)

Now, the increasing triangular map is unique given a fixed ordering. However, we do not know the ordering in practice. Thus consider the following idea: We stack autoregressive models with a ranodm permutation matrix in between. The determinant of a permutation matrix is just 1 so it does not affect computation.

11.6 Real NVP

Another parameterization is of the form

$$T_j(z_j; z_{<\ell}) = \begin{cases} z_j, & j < \ell \\ e^{\alpha_j(z_{<\ell})} z_j + \mu_j(z_{<\ell}), & j \geq \ell \end{cases}$$

In this (very) restricted parameterization, we can explicitly compute the inverse since the first block is the identity map, and the second block only depends on the first block. Recall that in general, we need to apply binary search.

11.7 Neural Autoregressive Flows (NAFs)

We parameterize

$$x_j = \mathrm{DNN}(z_j; w_j(z_{<j})) = T_j(z_j; z_{<j}).$$

Strictly positive weights and strictly monotonic activation functions ensure that the map is increasing.

11.8 Sum-of-Squares Polynomial Flows

The idea here is to approximate the non-decreasing derivative using sum of squares polynomials. Then, we integrate to get T.

11.9 Applications

11.9.1 Novelty Detection

Suppose we are given unlabeled data and we wish to detect novel data points.

Let U be uniform over $[0,1]^d$ and $X \in \mathbb{R}^d$ any random vector. Let $Q_X : [0,1]^d \to \mathbb{R}^d$ be an increasing triangular map such that $Q_X(U) \sim X$. Then we say Q_X is the *triangular quantile map*. Recall that this map is unique.

Moreover, we can stack increasing triangular maps and they remain increasing and triangular. Thus if $Y = T(X)$ for some increasing triangular map T, then $Q_Y = T \circ Q_X$.

Suppose $f_{\#}p$ is the data and q is a "standard" distribution (uniform, normal, etc). Then we solve

$$\min_{f,Q} \gamma \,\mathrm{KL}(f_{\#}p \| Q_{\#}q) + \lambda \ell(f) + \zeta g(Q)$$

where the 2nd and 3rd terms are for dimensionality reduction and regularization, respectively.

We paramterized A using SOS flow and solve with multiple gradient descent. There is no hyperparameter tuning needed!

Let $Z = f(X)$ be some dimensionality reduction. If the density

$$\log|Q'(Q^{-1}(Z))| + \frac{\|Q^{-1}(Z)\|^2}{2}$$

is too low, we consider it a novel point. Alternatively, if the quantile

$$\|Q^{-1}(Z) - \frac{1}{2}\|_{\infty}$$

is too low, we can also consider it a novel point.

11.10 Variational Autoencoders (VAE)

In this model, we essentially minimize the KL divergence of the joint density between the source and target distributions.

$$\min_{\theta} \min_{\phi} \mathrm{KL}\left[p(x)p_\theta(z \mid x) \| q(z)q_\theta(x \mid z)\right].$$

Interestingly, we can actually rewrite VAEs in terms of triangular flows.

Chapter 12

Optimal Transport

We are given a dataset and our goal is to estimate the density of the underlying distribution.

12.1 Definition

12.1.1 Motivation

Recall the stable matching problem where we are given as input a graph G with complete bipartition $V = (A, B)$. Each vertex $v \in V$ has a preference list of all neighbors for which they would like to be matched.

A matching is a subset of edges where the endpoints are disjoint. A *blocking pair* in a matching M is a pair of $uv, xy \in M$ such that uy, vx improves preferences of all parties.

We can also introduce a cost $c(i, j)$ for matching i with j. A *transferable cost* is one defined as

$$c(i, j) = i_j + j_i$$

where j is on the i_j-th position on the preference list of i.

Monge first formulated this as a minimum weight bipartite perfect matching problem

$$\min_{T:T \text{ is a bijection}} \sum_{i=1}^{n} c(i, T(i)).$$

12.1.2 Definition

In the continuous analogue, we have two distributions with densities p, q and we would like to solve

$$\min_{T\#p=q} \mathbb{E}\left[c(X, T(X))\right].$$

Here $X \sim p$ and $T_\# p$ is the push-forward distribution $T(X) \sim T_\# p$.

12.2 Solving the Problem

12.2.1 Relaxation

T is essentially a deterministic pairing. Let us instead relax the problem to a stochastic pairing

$$\min_{X\sim p, Y\sim q} \mathbb{E}\left[c(X, Y)\right] \le \min_{T_\# p=q} \mathbb{E}\left[c(X, T(X))\right].$$

We have $(X, Y) \sim \pi$ where the joint coupling π has marginals p, q. We replace the deterministic pairing $y = T(x)$ with a stochastic pairing $\pi(y \mid x)$. Surprisingly, at optimality $\pi(y \mid x)$ can be deterministic anyways.

12.2.2 Duality

Let us reconsider the optimization problem

$$\begin{aligned}
&\min_{X\sim p, Y\sim q} \mathbb{E}\left[c(X, Y)\right] \\
&= \min_{\pi\ge 0} \int c(x,y)\pi(x,y)dxdy && \int \pi(x,y)dy = p(x), \int \pi(x,y)dx = q(y) \\
&= \min_{\pi\ge 0} \max_{u(x),v(x)} \int \left[c(x,y) - u(x) - v(y)\right]\pi(x,y)dxdy \\
&\quad + \int u(x)p(x)dx + \int v(y)p(y)dy && \text{Lagrangian dual integrated} \\
&= \max_{u(x),v(x)} \min_{\pi\ge 0} \int \left[c(x,y) - u(x) - v(y)\right]\pi(x,y)dxdy \\
&\quad + \int u(x)p(x)dx + \int v(y)p(y)dy && \text{requires justification} \\
&= \max_{u(x),v(y)} \int u(x)p(x)dx + \int v(y)p(y)dy && u(x) + v(y) \le c(x,y) \\
&= \max_{u,v} \mathbb{E}[u(X)] + \mathbb{E}[v(Y)] && u(x) + v(y) \le c(x,y).
\end{aligned}$$

The second last equality is forced by the fact that if $u(x) + v(y) > c(x, y)$, then we can let $\pi \to \infty$ so that the objective is not finite.

12.2.3 Conjugacy

The condition $\forall x, y, u(x) + v(y) \leq c(x, y)$ implies that

$$\begin{aligned} u(x) &\leq \inf_y c(x, y) - v(y) \\ v(y) &\leq \inf_x c(x, y) - u(x). \end{aligned}$$

Thus we wish for

$$\begin{aligned} u(x) &= \inf_y c(x, y) - v(y) =: v^c(x) \\ v(y) &= \inf_x c(x, y) - u(x) =: u^c(y). \end{aligned}$$

12.2.4 Complementarity

Recall the equivalent optimization problem

$$\min_{\pi \geq 0} \max_{u(x), v(x)} \int [c(x, y) - u(x) - v(y)] \, \pi(x, y) dx dy \quad + \int u(x) p(x) dx + \int v(y) p(y) dy.$$

We determined that the first integral must be zero. Now, if $\pi(x, y) > 0$, then

$$\begin{aligned} u(x) + v(y) &= c(x, y) \\ u(x) &= \inf_z [c(x, z) - v(z)] =: v^c(x) \\ &\implies \\ y &\in \partial u(x) \qquad (\star) \\ x &\in \partial v(y) \\ \text{supp}(\pi) &\subseteq \text{graph}(\partial u) := \{(x, y) : y \geq \partial u(x)\}. \end{aligned}$$

$(\star)$ We understand that notation as $y \in \text{argmin}_z \, c(x, z) - v(z)$. Examples will make this concept more rigorous.

If ∂u is almost always a singleton, then $\pi(x, y) = \delta(x, \partial u(x))$.

12.2.5 Cyclic Monotonicity

Definition 12.2.1 (Cyclically Monotone)
A set $\Gamma \subseteq X \times Y$ is *C-cyclically monotone* if for any $k \in \mathbb{N}$ and $(x_1, y_1), \ldots, (x_k, y_k) \in \Gamma$, any cyclic permutation σ satisfies

$$\sum_{k=1}^{k} C(x_i, y_i) \leq \sum_{i=1}^{k} C(x_i, y_{\sigma(i)}).$$

Note that by taking $k = 2$, we arrive at the definition of having no blocking pairs!

Under certain conditions, $\text{supp}(\pi)$ is maximally cyclically monotone. In particular, when $\pi = (I \times \partial u)_{\#} p$, it means that the graph of ∂u is maximally cyclically monotone!

12.3 Applications

12.3.1 1-Wasserstein Distance

We wish to solve the problem

$$\min_{X \sim p, Y \sim q} \mathbb{E}\left[d(X, Y)\right] = \max_{u,v} \mathbb{E}[u(X)] + \mathbb{E}[v(Y)]$$
$$d(x, y) \geq u(x) + v(y).$$

Suppose v is 1-Lipchitz with respect to d. Thus

$$|v(y) - v(x)| \leq d(x, y)$$
$$\inf_y d(x, y) - v(y) \geq -v(x)$$

with equality when $x = y$.

The *Lipschitz envelope* of f is the largest Lipschitz function dominated by f. Specifically, $v^c(x) := \inf_y [d(x, y) - v(y)]$ is the Lipschitz envelope of $-v(x)$. We showed above that if v is already Lipschitz, then $v^c = -v$.

Wasserstein GAN

In the Wasserstein GAN, we restrict ourselves to Lipschitz functions u.

$$\min_{Z \sim p, Y \sim \hat{q}} \mathbb{E}\left[d(G(Z), Y)\right] = \max_{u \text{ Lipschitz}} \mathbb{E}[u(G(Z))] - \mathbb{E}[u(Y)].$$

Here $\hat{q}$ is the training set, p is the noise distribution, G is generator network, and u acts as the discriminator network.

Then we solve for G by optimizing

$$\min_G \max_{u \text{ Lipschitz}} \mathbb{E}[u(G(X))] - \hat{\mathbb{E}}[u(Y)].$$

12.3.2 2-Wasserstein Distance

Now consider

$$\min_{Z\sim p, Y\sim\hat{q}} \mathbb{E}\left[\frac{1}{2}\|X-Y\|_2^2\right] = \max_{u \text{ Lipschitz}} \mathbb{E}[u(X)] - \mathbb{E}[v(Y)]$$
$$\frac{1}{2}\|x-y\|_2^2 \geq u(x) + v(y).$$

Consider the function $v - \frac{1}{2}\|\cdot\|_2^2$. Then by taking the *concave conjugate*

$$v^c(x) := \inf_y \left[\frac{1}{2}\|x-y\|_2^2 - v(y)\right],$$

we set $u(x) = v^c(x)$.

Potential GAN

We now consider both u, v.

$$\min_{Z\sim p, Y\sim\hat{q}} \mathbb{E}\left[\frac{1}{2}\|G(Z)-Y\|_2^2\right] = \max_{u \text{ Lipschitz}} \mathbb{E}[u(G(Z))] - \mathbb{E}[u(Y)].$$

Here $\hat{q}$ is the training set, p is the noise distribution, G is generator network, and u, v act as the discriminator networks.

Then we solve for G by optimizing

$$\min_G \max_{u \text{ Lipschitz}} \mathbb{E}[u(G(X))] - \hat{\mathbb{E}}[v(Y)].$$

In practice, issues arrive since taking the conjugate of a neural network is non-trivial.

Potential Flow

Consider the function $T := \partial\left(\frac{1}{2}\|\cdot\|_2^2 - u\right)$. If p is any continuous distribution, then we have $T_{\#}p = q$. Thus $\text{supp}(\pi) = \text{graph}\, T$.

With this fact, we can attempt to solve the problem

$$\min_{u \text{ convex}} d((\partial u)\#p, q).$$

This can be parametrized by a ReLU network whose weights (excluding bias) are non-negative.

Now, both triangular and potential flows have a function $T : \mathbb{R}^d \to \mathbb{R}^d$ which pushes q to p. The function in triangular flows has a triangular Jacobian matrix, is preserved under composition, and fails rotational equivariance. On the other hand, the function in potential flows is maximally cyclically monotone (gradient of convex function), is not in general preserved under composition, and satisfies rotaional equivariance.

It is not hard to show that the two models are equivalent if and only if T is diagonal.

Chapter 13

Contrastive Estimation

13.1 Problem Formulation

Suppose we have a data density $x_1, \dots, x_n \sim p(x)$. We have a model density $q(x; \theta)$ with unknown parameter θ. The goal is to minimize some discrepancy score, say

$$\min_\theta D(p(x) \| q(x; \theta)).$$

13.2 Model Density

We have seen two of three popular parameterizations.

$$\begin{aligned}
q(x;\theta) &= \sum_k \pi_k q_k(x;\theta_k) && \text{mixture model} \\
q(x;\theta) &= T_\theta \# q_0(x) && \text{push-forward} \\
q(x;\theta) &= \exp(-E(x;\theta) - A(\theta)) && \text{energy} \\
A(\theta) &= \log \int \exp(-E(x;\theta)) dx.
\end{aligned}$$

Here q_0 is some noise distribution, say $q_0 \sim N(0, 1)$. The normalizing term $A(\theta)$ can be very difficult to compute. The first we have seen for GMMs and the second we have seen for GANs, triangular flows, and VAEs.

It is easy to sample, evaluate, and normalize mixture models. It is easy to sample, not in general possible to evaluate, but simple to normalize the push-forward model. It can be non-trivial to sample, difficult to fully evaluate, and non-trivial to normalize the energy model.

13.3 Discrepancy Measure

We have seen f-divergences, such as the KL-divergence, which amounts to MLE, as well as the Jenson-Shannon divergence, which leads to GANs. The is also the Wasserstein 1 and 2 distances from optimal transport. Finally, we can use RKHS to define *maximum mean discrepancy* (MMD).

We explore two more in detail: score matching and constrastive classification.

13.4 Constrastive Divergence

We wish to minimize

$$\min_\theta \mathrm{KL}(p(x) \| \exp[-E(x;\theta) - A(\theta)]).$$

We can simplify this optimization problem to

$$\min_\theta \int p(x) E(x;\theta) dx + A(\theta).$$

Upon taking the derivative of this objective function, we note that we can swap the order of the gradient and integral, under some conditions we omit.

$$\begin{aligned}
&\int p(x) \nabla_\theta E(x;\theta) dx + \nabla_\theta A(\theta) \\
&= \int p(x) \nabla_\theta E(x;\theta) dx - \int q(x;\theta) \nabla_\theta E(x;\theta) dx \\
&= \mathbb{E}_p[\nabla_\theta E(x;\theta)] - \mathbb{E}_q[\nabla_\theta E(x;\theta)].
\end{aligned}$$

While we cannot directly compute these expectations, we can estimate them using an empirical mean over a training set. We may need to use a Markov Chain Monte Carlo process for sampling.

13.5 Score Matching

Suppose we wish to minimize the *Fisher divergence*

$$\min_\theta \frac{1}{2} \mathbb{E}_p \| \nabla_x \log p(x) - \nabla_x \log q(x;\theta) \|_2^2.$$

Here $\nabla_x \log p(x)$ is known as the *score function*.

It is important that we take the log of our density before taking the gradient as this allows us to ignore the normalization term $A(\theta)$.

$$\begin{aligned}\boldsymbol{\nabla}_x \log q(x;\theta) &= \boldsymbol{\nabla}_x \log \exp[-E(x;\theta) - A(\theta)] \\ &= -\boldsymbol{\nabla}_x E(x;\theta).\end{aligned}$$

This allows us to simplify the objective to

$$\begin{aligned}&\min_\theta \int \langle \boldsymbol{\nabla}_x p(x), \boldsymbol{\nabla}_x E(x;\theta)\rangle dx + \frac{1}{2}\mathbb{E}_p \|\boldsymbol{\nabla}_x E(x;\theta)\|_2^2 \\ &= \min_\theta \langle \boldsymbol{\nabla}_x p(x), \boldsymbol{\nabla}_x E(x;\theta)\rangle + \frac{1}{2}\mathbb{E}_p \|\boldsymbol{\nabla}_x E(x;\theta)\|_2^2.\end{aligned}$$

Under some conditions, integration by parts justifies moving the gradient within the inner product to get

$$\begin{aligned}&\min_\theta \langle p(x), \Delta_x E(x;\theta)\rangle + \frac{1}{2}\mathbb{E}_p \|\boldsymbol{\nabla}_x E(x;\theta)\|_2^2 \\ &= \min_\theta \mathbb{E}_p \left[\Delta_x E(x;\theta) + \frac{1}{2}\|\boldsymbol{\nabla}_x E(x;\theta)\|_2^2\right]\end{aligned}$$

As usual, we can approximate the expectation with an empirical average over some training set.

13.6 Constrastive Classification

Suppose now that in addition to our data density $p(x)$ and our model density $q(x;\theta)$, we also have a background density $b(x)$, from which we can draw samples and ideally can describe explicitly.

Now, let us consider comparing densities ratios

$$\begin{aligned}\bar{p}(x) &:= \frac{p(x)}{b(x)} \\ \bar{q}(x) &:= \frac{q(x;\theta)}{b(x)}\end{aligned}$$

Now, these ratios are certainly non-negative but need not integrate to unity.

Then $b(x)\bar{q}(x;\theta)$ is an estimate of $p(x)$.

We minimize the following discrepancy

$$\min_\theta \mathbb{E}_b \left[-\bar{p}(x)\log \bar{q}(x;\theta) + (\bar{p}(x)+1)\log(\bar{q}(x;\theta)+1)\right].$$

This objective function is justified since if we set the derivative with respect to $\bar{q}(x;\theta)$ to 0, we see that

$$\frac{\bar{p}(x)}{\bar{q}(x;\theta)} = \frac{\bar{p}(x)+1}{\bar{q}(x;\theta)+1} \iff \bar{p}(x) = \bar{q}(x;\theta).$$

Expanding the expectation, we have

$$\begin{aligned}
&\int b(x)\left[-\frac{p(x)}{b(x)}\log\bar{q}(x;\theta) + \frac{p(x)+b(x)}{b(x)}\log(\bar{q}(x;\theta)+1)\right] \\
&= \mathbb{E}_p\left[-\log\frac{\bar{q}(x;\theta)}{\bar{q}(x;\theta)+1}\right] + \mathbb{E}_b\left[-\log\left(1-\frac{\bar{q}(x;\theta)}{\bar{q}(x;\theta)+1}\right)\right].
\end{aligned}$$

Once again, we would approximate these expectations using an empirical mean.

13.6.1 Interpretation as Classification

Introduce a label $y \in \{0,1\}$ where

$$x \mid y \sim b(x)^{1-y} \cdot p(x)^y.$$

We can then interpret $\bar{q}(x;\theta)$ as an odds ratio

$$\bar{q}(x;\theta) = \frac{\mathbb{P}\{y=1 \mid x\}}{\mathbb{P}\{y=0 \mid x\}} \iff \mathbb{P}\{y=1 \mid x\} = \frac{\bar{q}(x;\theta)}{\bar{q}(x;\theta)+1}.$$

The objective then reduces to logistic regression

$$\min_\theta \mathbb{E}_{x,y}\left[-y\log\mathbb{P}\{y=1 \mid x\} - (1-y)\log\mathbb{P}\{y=0 \mid x\}\right].$$

Part V

Safety

Chapter 14

Adversarial Robustness

14.1 Introduction

Suppose we have an image classification problem and we perturb the dataset by adding a small amount of noise. We would expect that our model predicts the same labels. However, this is not always the case!

This is an example of an adversarial input. Such instances are especially important when we rely on models for safety-critical functions like in autonomous vehicles.

Formally, suppose we have a fairly accurate classifier f such that $f(x) = y(x)$ on all points in our dataset. However, there exists some Δx, small in norm, such that $f(x + \Delta x) \neq y(x)$.

14.2 Huber's Loss

Consider *Huber's loss* given by

$$\begin{aligned} M^{\epsilon}_{|\cdot|}(t) &:= \min_s \frac{1}{2}(s-t)^2 + \eta|x| \\ &= \begin{cases} \frac{1}{2}t^2, & |t| \leq \eta \\ \eta|t| - \frac{1}{2}\eta^2, & |t| \geq \eta \end{cases} \\ M' &= \begin{cases} t, & |t| \leq \eta \\ \eta, & t \geq \eta \\ -\eta, & t \leq \eta \end{cases} \end{aligned}$$

Note the derivative is clipped. Thus when we train deep networks with gradient clipping, we are in some sense applying Huber's loss.

14.2.1 Huber's Contamination

Suppose the dataset comes from a mixture model

$$F = (1-\epsilon)G + \epsilon H$$

where G is the "true" distribution and H is some arbitrary contamination distribution.

We know ϵ but do NOT know which training points are noise. It can be shown that Huber's loss is in some sense asymptotically optimal in this setting.

14.3 Score Classifier

A *score function*, or classifier $f : X \to \mathbb{R}$ is L-Lipschitz continuous if

$$|f(x) - f(z)| \leq L\|x - z\|.$$

Remark this is equivalent to $\|\boldsymbol{\nabla} f\| \leq L$ when f is differentiable.

14.4 Attack Algorithms

Let L be some loss function, x_0 be the training example, y the true label, and $\bar{\theta}$ the trained parameter of the model. That is,

$$\bar{\theta} = \text{argmin}_\theta \, L(x_0, y; \theta).$$

In generating adversarial examples, we solve

$$\max_{x:\|x-x_0\|\leq\epsilon} L(x, y; \bar{\theta}).$$

14.4.1 Projected Gradient Method (PGM)

There are two popular algorithms for training attack algorithm. The PGM is given by

$$x \leftarrow x + \eta \cdot \boldsymbol{\nabla}_x L(x, y; \bar{\theta})$$
$$x \leftarrow \text{proj}(x).$$

14.4.2 Fast Gradient Sign Method (FGSM)

The FGSM is given by

$$x \leftarrow x + \epsilon \cdot \text{sign}(\boldsymbol{\nabla}_x L(x, y; \bar{\theta})).$$

Note the projection is implicit here

14.5 Targeted Attack

We can also try to force the model to predict a particular label y'.

$$\min_{x:\|x-x_0\|\leq\epsilon} L(x, y'; \bar{\theta}).$$

The algorithms for computing such examples are the similar.

14.6 Lipschitz Regularization

Let $W(F \parallel G)$ denote the Wasserstein distance between F, G

$$W(F \parallel G) := \max_{f:\mathrm{Lip}(f)\leq 1} \mathbb{E}_{X\sim F} f(X) - \mathbb{E}_{Z\sim G} f(Z).$$

We suppose now that G is the true distribution and consider the distributions which are not too far in W:

$$\mathcal{F}_\epsilon = \{F : W(F \parallel G) \leq \epsilon\}.$$

Under a linear model, we can consider the worst case of such distributions,

$$\begin{aligned}
&\min_w \max_{F\in\mathcal{F}_\epsilon} \mathbb{E}_{X\sim F} L(w^T X) \\
&= \min_{\gamma\geq 0} \gamma\epsilon^p - \mathbb{E}_{X\sim G} M^\gamma_{-\rho}(X; w) && \rho(x) := L(w^T x) \\
&\leq \mathbb{E}_{X\sim G} L(w^T x) + \epsilon\,\mathrm{Lip}(L_w).
\end{aligned}$$

Before, we justify the use of Huber's loss through the mixture model. Here, we use the Wasserstein distance to justify a Lipschitz regularization term.

14.7 Margin Story

Theorem 14.7.1
For almost all linearly separable binary datasets and any smoothly decreasing loss function with an exponential tail, gradient descent with small constant step size and any initial point w_0 converges to the unique solution $\hat{w}$ of hard-margin SVM, ie

$$\lim_{t\to\infty} \frac{w_t}{\|w_t\|} = \frac{\hat{w}}{\|\hat{w}\|}.$$

14.7.1 Binary Linear Classifiers

For a binary linear classifier which predicts positive if $w^T x > 0$, an adversarial example is given by

$$x = x_0 - \frac{x^T x_0}{\|w\|^2} w.$$

It has been hypothesized that as we train more and more, the training points are pushed towards the decision boundaries. This makes it easier to construct adversarial examples!

14.7.2 Deep Learning

We can think of a deep neural network as simultaneously learning a good representation as well as fitting a good linear model in that space. It is hypothesized that as we train these networks more and more, the learned representation pushes the training points towards the decision boundaries. Again, this makes it easier to construct aversarial examples.

14.8 Adversarial Training

We can attempt to minimize the impact of adversarial examples by changing the objective function with a min-max formulation

$$\min_w \mathbb{E}\left[\max_{\|\Delta x\| \leq \epsilon} L(x + \Delta x; w)\right].$$

The inner maximization can be solved by an attack algorithm.

In practice, we perform some gradient descent of the outer problem, then generate adversarial examples with an attack algorithm, then iterative.

14.9 Variational Loss

Theorem 14.9.1
All convex potential functions based boosters can not tolerate random classification noise at rate $\eta \in \left(0, \frac{1}{2}\right)$.

This motivates the use of non-convex loss functions

$$r(t) = \min_{0 \leq \eta \leq 1} \eta L(t) + \psi(\eta).$$

Note that we can perform alternating minimization to solve these loss functions.

14.10 Certification

There is interest in certifying whether an existing model is robust. Take the ball of radius ϵ about a point x. Let f be the model in question. We want to know if $f(B_\epsilon(x))$ has identical labels. Since the image can be highly non-convex, we can instead ask the same question about $\text{conv}(f(B_\epsilon(x)))$.

Chapter 15

Differential Privacy

Some typical performance metrics for machine learning include training time, memory, test time, and robustness. We would also like to avoid compromising user privacy since our models require access to potentially sensitive information.

15.1 Motivation

A first step towards privacy might be to anonymize the training set, ie remove all names from the training set. However, this has been shown to be insufficient! Indeed, with birth date and full postal code, one can identify 97% of voters!

Alternatively, we might require "large" queries to the dataset instead of giving fine-grained access. This can also be exploited by a sequence of adversarial queries.

We might also think about refusing to provide and information. The obvious downside of this approach is that machine learning is no longer possible. Moreover, this still yields some information in some cases! For instance, refusing to answer "Did you cheat on the exam?"

Randomization is key! Consider the following scheme. We ask each interviewee to toss a fair coin in private. If it is heads, answer honestly. Otherwise, answer randomly. We see that cheaters say "yes" with probability $\frac{3}{4}$, while non-cheaters answer "yes" with probability $\frac{1}{4}$. The total fraction of "yes" should be

$$\frac{3}{4}p + \frac{1}{4}(1-p),$$

from which we can estimate p. Note that this scheme offers plausible deniability for each individual!

15.2 Differential Privacy

Definition 15.2.1 (Differential Privacy)
We say that a randomized algorithm $\mathcal{M}$ is (ϵ, δ)-*DP* if for all $S \subseteq \text{Im}(\mathcal{M})$ and for all neighboring datasets D_1, D_2,

$$\mathbb{P}\{\mathcal{M}(D_1) \in S\} \leq \exp(\epsilon) \cdot \mathbb{P}\{\mathcal{M}(D_2) \in S\} + \delta.$$

If $\delta = 0$, we say that $\mathcal{M}$ is ϵ-*DP*. Then we can interpret

$$\epsilon \geq \left| \log \frac{\mathbb{P}\{\mathcal{M}(D_1) \in S\}}{\mathbb{P}\{\mathcal{M}(D_2) \in S\}} \right|.$$

Example 15.2.1
Randomized response is $(\ln 3, 0)$-DP.

Note that if D, D' differ by k entries, then an ϵ-DP algorithm yields

$$\mathbb{P}\{\mathcal{M}(D) \in S\} \leq \exp(k\epsilon)\mathbb{P}\{\mathcal{M}(P_2) \in S\}.$$

Proposition 15.2.2
Let $\mathcal{M} : \mathscr{D} \to \mathscr{E}$ be (ϵ, δ)-DP and $\mathcal{N} : \mathscr{D} \to \mathscr{F}$ be any randomized mapping independent of $\mathcal{M}$. Then the composition $\mathcal{N} \circ \mathcal{M}$ is also (ϵ, δ)-DP.

Proof
Indeed,

$$\mathbb{P}\{\mathcal{N}(\mathcal{M}(D)) \in S\} = \mathbb{P}\{\mathcal{M}(D) \in \mathcal{N}^{-1}(S)\}.$$

15.2.1 Laplacian Mechanism

Theorem 15.2.3
Given any function $f : \mathscr{D} \to \mathbb{R}^d$, the Laplacian mechanism returns

$$\mathcal{M}_L(D, f, \epsilon) = f(D) + Z$$

where Z_j are iid Laplacian random variables with

$$\lambda = \frac{\|\boldsymbol{\nabla} f\|_1}{\epsilon}.$$

The Laplacian mechanism achieves ϵ-DP.

15.2.2 Calculus of DP

Proposition 15.2.4
Let $\mathcal{M}_i : \mathscr{D}_i \to \mathscr{E}_i$ be (ϵ_i, δ_i)-DP. Then the product map

$$\mathcal{M} : \prod \mathscr{D}_i \to \prod \mathscr{E}_i$$

is (ϵ, δ)-DP for

$$\epsilon := \sum_i \epsilon_i$$
$$\delta := \sum_i \delta_i.$$

15.3 Influence Function

Definition 15.3.1 (Influence Function)
Let $T : \mathscr{D} \to \mathbb{R}$ be a statistic. Its influence function at distribution $F \in \mathscr{D}$ and point $x \in \mathbb{R}^d$ is

$$\mathrm{IF}(T, F, x) := \lim_{t \to 0} \frac{T\left[(1-t)F + t\delta_x\right] - T(F)}{t}.$$

$\|\mathrm{IF}(T, F, \cdot)\|_\infty$ is a measure of robustness and privacy.

Example 15.3.1
The influence of the mean function is infinite.

Chapter 16

Abstention

Our classifiers are not always certain about their predictions. Thus in some cases, it might make sense to allow our models to abstain from making a prediction.

16.1 Baye's Rule

We attach a cost for each prediction $\Delta(y, \hat{y})$ where $y \in [c]$ is true label and $\hat{y} \in [c] \cup \{0\}$ is the prediction. $\hat{y} = 0$ means abstain.

> **Example 16.1.1 (Constant Cost)**
> A simple example is the following.
>
> $$\Delta_\lambda(y, \hat{y}) = \begin{cases} 0, & \hat{y} = y \\ 1, & 0 \neq \hat{y} \neq y \\ \lambda, & \hat{y} = 0 \end{cases}$$
>
> where λ is the abstention cost. We will see that $\lambda \in [0, 1 - 1/c]$ or else the model always abstains or never abstains.

We wish to solve the minimization problem

$$\inf_{\hat{y}: \mathcal{X} \to \{0, \dots, c\}} \mathbb{E}\Delta_\lambda(Y, \hat{y}).$$

Recall the law of total expectation

$$\mathbb{E}_X \left[\mathbb{E}_Y \{ f(X, Y \mid X = x) \} \right].$$

Fixing $X = x$,

$$\int_{\hat{y}\in\{0,\dots,c\}} \mathbb{E}_Y \left[\Delta_\lambda(Y,\hat{y}) \mid X = x\right] = \begin{cases} \lambda, & \hat{y} = 0 \\ \mathbb{P}\{Y \neq \hat{y}\}, & \hat{y} \neq 0 \end{cases}$$

Define MAP probability

$$\bar{p}(x) := \max_{k\in[c]} \mathbb{P}\{Y = k \mid X = x\}.$$

The optimal Baye's rule is given by

$$\hat{y}(x) = \begin{cases} 0, & \bar{p}(x) < 1 - \lambda \\ k, & \mathbb{P}\{Y = k \mid X = x\} = \bar{p}(x) \geq 1 - \lambda \end{cases}$$

For any abstention cost λ, we predict if $\bar{p}(x) \geq 1 - \lambda$ and abstain otherwise. This shows that we should choose $\lambda < 1 - \bar{p}(x)$ which is bounded above by $1 - \frac{1}{c}$.

The larger the abstention cost, it is less likely to abstain. Also note that we need only estimate $\bar{p}(x)$ once and apply to any abstention cost λ.

16.1.1 Error-Reject Trade-Off

We can also decompose the classification risk as the error and rejection

$$\mathbb{E}\left[\Delta_\lambda(Y,\hat{y})\right] = \mathbb{P}\{Y \neq \hat{y} \neq 0\} + \lambda\mathbb{P}\{\hat{y} = 0\}.$$

16.2 Binary Surrogate Risk Minimization

Suppose now we are in the binary setting with classes $\{-1, 1\}$.

Let us introduce a differentiable loss function ℓ and instead minimize the risk

$$\min_{\hat{y}:\mathcal{X}\to\mathbb{R}} \mathbb{E}\ell(Y, \hat{y}(X)).$$

Conditioning on $X = x$ and introducing $\eta(x) = \mathbb{P}\{Y = 1 \mid X = x\}$, this simplies to

$$\min_{\hat{y}(x)\in\mathbb{R}} \eta\ell(\hat{y}) + [1 - \eta]\ell(-\hat{y}).$$

The derivative is zero if and only if

$$\frac{\eta}{1-\eta} = \frac{\ell'(-\hat{y})}{\ell'(\hat{y})} =: s(\hat{y}).$$

Now, the optimal Bayes rule previously required access to $\eta(x)$. Indeed, we would predict 1 if $\eta(x) \geq 1 - \lambda$, -1 if $\eta(x) \leq \lambda$, and abstain otherwise. But this is equivalent to

$$\hat{y}_\lambda^\star(x) = \begin{cases} 1, & s(\hat{y}) \geq \frac{1-\lambda}{\lambda} \\ -1, & 0 \leq s(\hat{y}) \leq \frac{\lambda}{1-\lambda} \\ 0, & \text{else} \end{cases}$$

Note that we have removed the dependence on $\eta(x)$.

$s(\hat{y})$ can be computed for common loss functions such as logistic loss, exponential loss, and squared hinge loss. In all cases, just train the classifier as usual but compare the thresholds for prediction with some abstention cost!

For hinged loss, the straightforward method does not work. We can introduce some abdaptive hinge loss which depends on λ. Unfortunately, this means that training will depend on the cost.

Chapter 17

Causality

The overall goal of causal inference, like much of statistics, is to make sense of data, guide actions and policies, as well as learn from successes and failures. For instance, we would like to estimate the effect of a cause, or identify the cause of an effect. A central question is how and why do causes influence their effects?

Conventional statistics and machine learning that aim to describe data is insufficient. We need to understand how data is generated. Much of the popular techniques focus on correlation instead of causation. When we have this stronger goal, the golden standard of randomized control experiments are not always application and we have to deal with (passive) observational data.

17.1 Motivation

17.1.1 The Prosecutor's Fallacy

Consider the case of OJ Simpson, who was prosecuted for the potential murder of his wife. His attorney argued that "only one in a thousand abusive husbands eventually murder their wives." We will see that this argument is misleading!

Let G be the event that a husband murdered his wife, B the event that a husband abuses his wife, and M the event that the wife is murdered. The attorney's argument is that

$\mathbb{P}\{G \mid B\} = \frac{1}{1000}$. However, the true probability we should have considered is

$$\begin{aligned}\mathbb{P}\{G \mid B, M\} &= \frac{\mathbb{P}\{G, M \mid B\}}{\mathbb{P}(G, M \mid B) + \mathbb{P}(\bar{G}, M \mid B)} \\ &= \frac{\frac{1}{1000}}{\frac{1}{1000} + \frac{999}{1000}\epsilon} \\ &\approx 1.\end{aligned}$$

17.1.2 Exercise is Bad?

From a dataset of hours of exercise and cholesterol level, we can observe correlation between the amount of exercise and cholesterol level. This seems to suggest we should not exercise!

The explanation for this phenomenon is the confounding variable of age. Older folks tend to have higher cholesterol and also exercise more.

17.1.3 Simpson's Paradox

The statistician Simpson noticed that a drug benefits the recovery of both men and women. However, when considered as a whole population, it seemed to have negative effect overall!

This seems very unusual but can still be explained by a counfounding variable. It can be seen from data that women are more likely to take the drug. If we assume that women are less likely to recover regardless of the drug, we see that a random drug user is less likely to recover than a random non-drug user.

In other words, being a woman is a common cause for both drug taking and failure to recover. Under this assumption of causal mechanism, it makes sense to consult segregated data in order to single out the effect of the drug.

On the other hand, suppose we segregate the data by post-treatment blood pressure and swap the numbers for drug and no drug. We see that the segregated data shows negative effects for recovery while there is a positive effect as a whole.

Suppose the drug lowers blood pressure which helps recovery but also has some toxic side effect. If we hold the blood pressure fixed, we can only observe the toxicity. In this case, it makes more sense to consult the data as a whole.

The two situations involve the exact same dataset (ignoring labels). What changed is the causal mechanism behind the data generation. In order to account for these, we need more techniques than just the data itself.

17.2 Introduction to Causality

17.2.1 Structural Causal Models (SCM)

In a SCM, our setting is a DAG $G = (V, E)$. Our primary interest is in the *endogenous* random variables $X_1, \dots, X_n \subseteq V$. We say that X_i is a *direct cause* of X_j if $X_i \in p(X_j)$ is a parent of X_j. Similarly, we say X_i is a *cause* of X_i if $X_i \in a(X_j)$ is an ancestor. We also have some *exogenous* random variables $U_1, \dots, U_m$.

We enforce that exogenous variables are not descendants of any other variable. On the other hand, endogenous variables must be descendants of at least 1 exogenous variable and possibly other endogenous ones.

Finally, we assume a functional relation

$$X_i = f_i(p(X_i); U).$$

Example 17.2.1
Typically, we have $m = n$ and exogenous variable U_i is a direct cause of endogenous variable X_i. Thus we have

$$X_i = f_i(p(X_i); U_i).$$

17.2.2 Bayesian Networks

We say a joint distribution over $X_1, \dots, X_n$ *factorizes over a DAG G (with nodes X_i)* if

$$\begin{aligned}\mathbb{P}(X_1, \dots, X_n) &= \prod_{i=1}^{n} \mathbb{P}\{X_i \mid X_1, \dots, X_{i-1}\} \\ &= \prod_{i=1}^{n} \mathbb{P}\{X_i \mid p(X_i)\}.\end{aligned}$$

A *chain* DAG is a directed path.

Theorem 17.2.2
Two variables X, Y are conditionally independent given a set of variables $\mathcal{Z}$ if there is only one path between X, Y and the path is unidirectional and intercepts Z.

Theorem 17.2.3
Two variables X, Y are conditionally independent given another variable Z if there is only one path between X, Y and Z is a common cause of X, Y.

Theorem 17.2.4
Two variables X, Y are independent but conditionally dependent given another variable Z if there is only one path between X, Y and Z is a common effect (descendant) of X, Y.

We can distill the three theorems into a unified one.

Definition 17.2.1 (Blocked)
A path between two nodes X, Y is blocked by a set of nodes $\mathcal{Z}$ if any of the below holds:

(a) The path contains a chain $A \to B \to C$ where $B \in \mathcal{Z}$

(b) The path contains a fork $B \to A, C$ where $B \in \mathcal{Z}$

(c) The path contains a v-structure $A, C \to B$ where no descendants of B is in $\mathcal{Z}$.

Theorem 17.2.5
If $\mathcal{Z}$ blocks every path from any node of $\mathcal{X}$ to any node of $\mathcal{Y}$, then conditional on $\mathcal{Z}$, $\mathcal{X}, \mathcal{Y}$ are d-separated and hence independent.

We can also reduce the case of directed graphs to undirected graphs.

Definition 17.2.2 (Moralization)
The moralization of a DAG G is the undirected graph H obtained by G by adding an edge between all co-parents and taking the underlying graph.

Theorem 17.2.6
Two disjoint sets $\mathcal{X}, \mathcal{Y}$ are independent conditional on $\mathcal{Z}$ if every path from $\mathcal{X} \to \mathcal{Y}$ passes through $\mathcal{Z}$ in the moralization of the graph induced by ancesters of $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$.

$$G[a(\mathcal{X}, \mathcal{Y}, \mathcal{Z})].$$

17.2.3 Testing Causal Models

An assumed causal model implies may conditional independences. If any of these relations are not supported by data, we can reject the causal model. In fact, a failed conditional independence indicates how to repair the causal model. If all relations hold, then the causal model cannot be refuted based on purely observational data.

Theorem 17.2.7
Two causal graphs imply the same set of conditional independences if they share the same underlying undirected graph and they share common v-structures.

Through these causal models, we can perform studies such as conditioning, intervention, and counterfactual analysis.

Chapter 18

Interpretability

18.1 Activation Maximization

Our goal is to understand a neuron activation. We can do so by fixing the model weights and running (constrainted) gradient ascent on the input.

18.2 Coalition

We formalize interpretability as a cooperative game where each feature, training example, and neuron is a player. The payoff of a coalition is the performance metric (accuracy).

What is the value $\psi_i(v)$ of player i?

18.2.1 Shapley's Axioms

If $v(S \cup \{i\}) = v(S \cup \{j\})$ for all $S \not\ni i, j$, then $\psi_i(v) = \psi_j(v)$ (symmetry).

If $v(S \cup \{i\}) = v(S)$ for all S, then $\psi_i(v) = 0$ (dummy).

$\sum_{i \in N} \psi_i(v) = v(N)$ (efficiency).

$\psi_i(v + u) = \psi_i(v) + \psi_i(u)$ (additivity).

> **Example 18.2.1**
> Fix some $T \subseteq N$ and consider $v_T(S) := c_T \mathbb{1}\{T \subseteq S\}$. We can show that the axioms imply that $\psi_i(v_T) = \frac{c_T}{|T|}$ for $i \in T$ and 0 otherwise.

It can be shown that any characteristic function $v : 2^N \to \mathbb{R}$ can be written as

$$v = \sum_{T \neq \varnothing} c_T v_T.$$

It follows that

$$\begin{aligned}\psi_i(v) &= \sum_{T \neq \varnothing} \psi_i(c_T v_T) \\ &= \sum_{N \supseteq T \ni i} \frac{c_T}{|T|}.\end{aligned}$$

Alternatively, we can write

$$\psi_i(v) = \mathbb{E}_\pi \phi_i(v, \pi)$$

where

$$\phi_i(v, \pi) = n(\pi(1), \ldots, \pi(k)) - v(\pi(1), \ldots, \pi(k-1))$$

where $\pi(k) = i$. More explicitly,

$$\psi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} [v(S \cup \{i\}) - v(S)].$$

18.3 Counterfactual

$$\begin{aligned}w_\epsilon &:= \operatorname{argmin}_w \bar{\ell}(w) + \epsilon f(w) \\ \frac{dw_\epsilon}{d_\epsilon}(0) &= -\left[\nabla^2 \bar{\ell}(w_0)\right]^{-1} \dot{\nabla} f(w_0) \\ \frac{d\ell(w_\epsilon; x)}{d\epsilon}(0) &= \nabla_w^T \ell(w_0; x) \cdot \frac{dw_\epsilon}{d\epsilon}(0) \\ &= -\nabla_w^T \ell(w_0; x) \left[\nabla^2 \bar{\ell}(w_0)\right]^{-1} \cdot \nabla f(w_0).\end{aligned}$$

We can compute the change in training loss by a small additive perturbation $\epsilon f(w)$ of the weights.

This model allows us to ask what happens to the training loss when we remove a single training point for example.

18.3.1 Dataset Poisoning

This model also allows us to ask what happens to the training loss when we perturb a single training point.