CMPUT 567: Machine Learning 2

Keven Qiu
Instructor: Martha White

Contents

1 Generalization Error

The generalization error for a given f, with squared error is

$$E_{x,y}[(f(X) - Y)^2] = \underbrace{E[(f(X) - f^*(X))^2]}_{\text{reducible error}} + \underbrace{E[(f^*(X) - Y)^2]}_{\text{irreducible error}}$$

where $f^*(x) = E[Y|x]$ and the irreducible error is $E_x[\text{Var}(Y|X)]$.

Bias and variance is about the function class $\mathcal{F}$ (and algorithm, i.e. objective and optimizer).

$$f_{\mathcal{D}} = \text{Alg}(\mathcal{D})$$

where $\mathcal{D} \sim \underbrace{p(x_1, y_1)p(x_2, y_2) \cdots p(x_n, y_n)}_{p(\mathcal{D})}$. Then

$$E_{\mathcal{D}}[GE(f_{\mathcal{D}})] = \underbrace{E_{\mathcal{D}}[\text{reducible error } (f_{\mathcal{D}})]}_{\text{bias}^2 + \text{variance}} + \text{irreducible error}$$

Note there are two variances: the irreducible error and the variance of our function f.

For one x:

$$\begin{aligned} E_{\mathcal{D}}[(f_{\mathcal{D}}(x) - f^*(x))^2] &= E_{\mathcal{D}}[(\underbrace{f_{\mathcal{D}}(x) - E_{\mathcal{D}}[f_{\mathcal{D}}(x)]}_{a} + \underbrace{E_{\mathcal{D}}[f_{\mathcal{D}}(x)] - f^*(x)}_{b})^2] \\ &= \underbrace{b^2}_{\text{bias}^2} + \underbrace{a^2}_{\text{variance}} + 2\underbrace{E_{\mathcal{D}}[f_{\mathcal{D}}(x) - E_{\mathcal{D}}[f_{\mathcal{D}}(x)]]}_{=0} b \\ &= (E_{\mathcal{D}}[f_{\mathcal{D}}(x)] - f^*(x))^2 + \underset{\mathcal{D}}{\text{Var}}(f_{\mathcal{D}}(x)) \end{aligned}$$

If we can represent f^* (high model complexity), then $E_{\mathcal{D}}[f_{\mathcal{D}}(x)] \approx f^*(x)$ for all x.

1.1 Bias and Variance for Linear Regression

With nonlinear representation Φ, we have by SVD

$$\Phi = U\Sigma V^T \in \mathbb{R}^{n \times p}$$

where $U = [u_1, \ldots, u_n]$, with left singular vectors $u_j \in \mathbb{R}^{n\times 1}$ and $V = [v_1, \ldots, v_p]$, with right singular vectors $v_j \in \mathbb{R}^{p\times 1}$, and $\Sigma \in \mathbb{R}^{n\times p}$.

We consider thin SVD, where we consider only the first p rows/columns.

$$\Sigma = \begin{bmatrix} \sigma_1 & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \sigma_p \\ \vdots & \vdots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}, \Sigma_p = diag(\sigma_1, \ldots, \sigma_p), U_p = [u_1, \ldots, u_p]$$

since $U\Sigma = U_p\Sigma_p$. Note that $\Sigma^T\Sigma = \Sigma_p^2$.

$$\begin{aligned} \Phi^T\Phi &= V\Sigma^T U^T U\Sigma V^T \\ &= V\Sigma^T\Sigma V^T \\ &= V\Sigma_p^2 V^T \\ \mathbf{w} &= (\Phi^T\Phi)^{-1}\Phi^T y \\ &= V\Sigma_p^{-2}V^T V\Sigma_p U_p^T y \\ &= V\Sigma_p^{-1}U_p^T y \\ &= \sum_{j=1}^{p} \frac{u_j^T y}{\sigma_j} v_j \end{aligned}$$

Note that for this, all singular values must be nonzero/positive. $\Phi^t = V\Sigma_p^{-1}U_p^T$ is known as the pseudoinverse of Φ.

If Φ has lower rank $r < p$, i.e. $\sigma_1, \ldots, \sigma_r > 0, \sigma_{r+1}, \ldots, \sigma_p = 0$, then $\Phi^t = V_r\Sigma_r^{-1}U_r^T$ where $V_r = [v_1, \ldots, v_r]$.

Notice that $\mathbf{w}$ is sensitive to small singular values since we divide by them. Reasons why σ_j can be small:

- linearly correlated features (one dimension might not have anything)
- insufficient data

1.2 ℓ_2-Regularization

For linear regression, the MLE is $p(y|x) = \mathcal{N}(\phi w, \sigma^2)$. MAP with Gaussian prior is $-\ln p(w_j) =$ constant $+ \frac{\lambda}{2\sigma^2}w_j^2$.

$$c(w) = \frac{1}{2}\|\Phi w - y\|_2^2 + \frac{\lambda}{2}||w||_2^2 \implies (\Phi^T\Phi + \lambda I)w = \Phi^T y \implies w = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T y$$

Using SVD $\Phi^T\Phi + \lambda I = V(\Sigma_p^2 + \lambda I)V^T$,

$$w_{MAP} = V(\Sigma_p^2 + \lambda I)^{-1}U_p^T y = \sum_{j=1}^{p} \frac{\sigma_j}{\sigma_j^2 + \lambda}(u_j^T y)v_j$$

As long as large λ, the fraction does not blow up like it did without ℓ_2-regularization.

1.3 Better Generalization Error

Realizable setting: $y \sim \mathcal{N}(\Phi w^*, \sigma^2)$ for some w^* and $f^* \in \mathcal{F}$ (f^* is in function class).

Let $w(\mathcal{D})$ be our estimator.

$$MSE(w(\mathcal{D})) = E_{\mathcal{D}}\left[\sum_{j=1}^{p}(w_j(\mathcal{D}) - w_j^*)^2\right] = \sum_{j=1}^{p}(E[w_j(\mathcal{D})] - w_j)^2 + \sum_{j=1}^{p}\text{Var}(w_j(\mathcal{D}))$$

Notice $y = \phi w^* + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. For MLE,

$$\begin{aligned} E[w_{MLE}(\mathcal{D})] &= E[(\Phi^T\Phi)^{-1}\Phi^T y] \\ &= E[(\Phi^T\Phi)^{-1}\Phi^T(\Phi w^* + \varepsilon)] \\ &= E[(\Phi^T\Phi)^{-1}(\Phi^T\Phi)w^*] + \underbrace{E[(\Phi^T\Phi)^{-1}\Phi^T]}_{=0}E[\varepsilon] \\ &= w^* \end{aligned}$$

This is unbiased as it equals w^*. For MLE variance,

$$\sum_{j=1}^{p}\text{Var}(w_{MLE}(\mathcal{D})) = \sigma^2 E\left[\sum_{j=1}^{p}\sigma_j^{-2}\right]$$

where first σ^2 is variance, not singular value. This can be very big if σ_p is often small.

For MAP,

$$E[w_{MAP}(\mathcal{D})] = E[(\Phi^T\Phi + \lambda I)^{-1}\Phi^T(\Phi w^* + \varepsilon)] \neq w^*$$

so this is biased as λ increases.

$$\sum_{j=1}^{p}\text{Var}(w_{MAP}(\mathcal{D})) = \sigma^2 E\left[\sum_{j=1}^{p}\frac{\sigma_j^2}{(\sigma_j^2 + \lambda)^2}\right]$$

This is not very big if λ is large.

Non-realizable: $y = \mathcal{N}(f^*(x), \sigma^2)$, but $f^* \notin \mathcal{F}$. w_{MLE} is also biased.

Note about n: We can stick a $\frac{1}{n}$ in front of $\Phi^T\Phi$ to act like an average over outerproducts. Note $\frac{1}{n}\Phi^T\Phi = E[\Phi^T\Phi]$.

Without normalization, $\sigma_j^2 \to \infty$ as $n \to \infty$. With normalization $\frac{1}{n}\sigma_j^2 \to$ singular values for $E[\Phi^T\Phi]$.

$\sum_{j=1}^{p}\sigma_j^{-2} \to 0$ as $n \to \infty$, so $\text{Var}(w_{MLE}(\mathcal{D})) \to 0$ as $n \to \infty$.

Multiple outputs: $y \in \mathbb{R}^m, \phi(x)W \approx y$.

$$Y = \begin{bmatrix} y_{11} & \cdots & y_{1m} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{nm} \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1 & \cdots & \mathbf{y}_m \end{bmatrix}$$

where $\mathbf{y}_i \in \mathbb{R}^n$. So

$$\mathbf{w} = \Phi^t Y = \begin{bmatrix} \Phi^t\mathbf{y}_1 & \cdots & \Phi^t\mathbf{y}_m \end{bmatrix}$$

2 Optimization

Find

$$\min_{w \in \mathbb{R}^p} c(w)$$

For example, $c(w) = \frac{1}{n}\sum_{i=1}^{n} c_i(w)$, linear regression $c_i(w) = \frac{1}{2}(\phi(x_i)w - y_i)^2$ or logistic regression $c_i(w) = -y_i \ln f_w(x_i) - (1-y_i)\ln(1 - f_w(x_i))$.

1. Find a stationary point w_0, $\nabla c(w_0) = 0$.
2. Check type using second derivative test ($c_i'(w_t) < 0, > 0, = 0$).

In p-dimensions, the **Hessian** matrix $H_{c(w_t)} \in \mathbb{R}^{p \times p}$, where

$$H_{c(w)}[i,j] = \frac{\partial^2 c}{\partial w_i \partial w_j}(w)$$

E.g. $c(w) = \frac{1}{2}(w_1 + xw_2 - y)^2$, $\nabla c(w) = \begin{bmatrix} (w_1 + xw_2 - y) \\ (w_1 + xw_2 - y)x \end{bmatrix} = \begin{bmatrix} \frac{\partial c}{\partial w_1}(w) \\ \frac{\partial c}{\partial w_2}(w) \end{bmatrix}$.

$$H_{c(w)} = \begin{bmatrix} \frac{\partial}{\partial w_1}\frac{\partial}{\partial w_1}c(w) & \frac{\partial}{\partial w_1}\frac{\partial}{\partial w_2}c(w) \\ \frac{\partial}{\partial w_2}\frac{\partial}{\partial w_1}c(w) & \frac{\partial}{\partial w_2}\frac{\partial}{\partial w_2}c(w) \end{bmatrix} = \begin{bmatrix} 1 & x \\ x & x^2 \end{bmatrix}$$

For $(x = 0.1, y = 2)$ and $w_0 = (0,0)$,

$$\nabla c(w) = \begin{bmatrix} -2 \\ -0.2 \end{bmatrix}, H_{c(w_0)} = \begin{bmatrix} 1 & 0.1 \\ 0.1 & 0.01 \end{bmatrix}$$

The eigenvalues of $H_{c(w_0)}$ are $\lambda_1 \approx 1, \lambda_2 = 0$. So we get a concave up in one direction, and flat in the other direction (a taco shape).

E.g. $c(w) = \frac{1}{2}(w_1 + x_1 w_2 - y_1)^2 + \frac{1}{2}(w_1 + x_2 w_2 - y_2)^2$. At $(x_1 = 0.1, y_1 = 2)$ and $(x_2 = -0.3, y_2 = -1)$,

$$H_{c(w_0)} = \begin{bmatrix} 2 & -0.2 \\ -0.2 & 0.1 \end{bmatrix}$$

for $w_0 = (0,0)$. The eigenvalues are $\lambda_1 = 6.5, \lambda_2 = 0.08$.

Better Steps: $H_{c(w)}^{-1}$ is a matrix stepsize. The second-order method is

$$w_{t+1} = w_t - H_{c(w_t)}^{-1} \nabla c(w_t)$$

The scalar version was $w_{t+1} = w_t - \frac{1}{c''(w_t)} \cdot c'(w_t)$.

Using eigenvalue decomposition, $H_{c(w_t)}^{-1} \nabla c(w_t) = U \Lambda^{-1} \underbrace{U^T \nabla c(w_t)}_{\tilde{g}_t}$. This shows that the step is inversely proportional to curvature magnitude (λ_i's). One step size could be bad, since in regular gradient descent uses stepsize η, so Hessian allows us not to step too much on step surfaces and more on flat surfaces.

Computational issues: Computing $H_{c(w)}^{-1}$ is $O(p^2 n)$ for linear regression. This is because $H_{c(w)} = \frac{1}{n}\sum_{i=1}^{n} H_{c_i(w)}$, where $H_{c_i(w)}$ takes $O(p^2)$ and you sum n times. Computing the inverse is $O(p^3)$, but for $p < n$, $p^3 < p^2 n$, so $O(p^2 n)$ overall.

An alternative is to use a vector of stepsizes $\eta_t \in \mathbb{R}^p$. So

$$H_{c(w_t)}^{-1} \approx diag(\eta_{t,1}, \eta_{t,2}, \ldots, \eta_{t,p})$$

and the update is

$$w_{t+1} = w_t - \eta_t \overbrace{\cdot}^{\text{element-wise product}} \nabla c(w_t)$$

We do not use a fixed $\eta \in \mathbb{R}^p$. We use vector stepsizes, that change with time.

Definition 2.1: Adagrad

$$\eta_{t,j} = (1 + \overline{g}_{t,j})^{-1/2}$$

where $\overline{g}_{t,j} = \overline{g}_{t-1,j} + g_{t,j}^2$ and $\overline{g}_0 = 0$.

This has nice theory, but in practice, stepsizes shrink too much since we have a monotonic decrease in stepsize.

An intuitive modification is to take an exponential average of $g_{t,j}^2$, which leads to RMSProp.

Definition 2.2: RMSProp

$$v_{t,j} = (1-\beta) g_{t,j}^2 + \beta v_{t-1,j}, \beta > 0$$
$$\eta_{t,j} = \frac{\eta}{\sqrt{v_{t,j} + \varepsilon}} = \eta (v_{t,j} + \varepsilon)^{-1/2}$$

Also add **momentum**: with fixed stepsize η,

$$w_{t+1} = w_t - \eta m_{t+1}, m_{t+1} = g_t + \beta_2 m_t, \beta_2 > 0$$

An alternate form is

$$w_{t+1} = w_t - \eta g_t + \beta(w_t - w_{t-1})$$

Definition 2.3: Adam

Puts RMSProp and momentum together with extra modification.

There is a bias correct for $v_{0,j} = 0$. There is an exponential average in momentum $m_{t+1} = (1 - \beta_2) g_t + \beta_2 m_t$. The last modification is

$$\text{Adam's } \frac{1}{\sqrt{v_{t,j}} + \varepsilon} \text{ vs. RMSProp's } \frac{1}{\sqrt{v_{t,j} + \varepsilon}}$$

2.1 Comparisons

Second-order updates takes $O(p^2 n)$ per iteration and gradient descent takes $O(pn)$ per iteration.

Definition 2.4: Stochastic Gradient Descent (SGD)

Randomly pick $\mathcal{B}_t \subseteq \{1, \ldots, n\}$ with $|\mathcal{B}_t| = b$, the gradient $\hat{g}_t = \frac{1}{b} \sum_{i \in \mathcal{B}_t} \nabla c_i(w_t)$,

$$w_{t+1} = w_t - \eta_t \hat{g}_t = w_t - \underbrace{\eta_t \nabla c(w_t)}_{\text{GD}} + \eta_t \underbrace{\left(\nabla c(w_t) - \overbrace{\frac{1}{b} \sum_{i \in \mathcal{B}} \nabla c_i(w_t)}^{\hat{g}_t} \right)}_{\text{noise term, expected value=0}}$$

The runtime is $O(pb)$.

This is much faster than $O(pn)$ for GD. $\hat{g}_t \approx g_t = \frac{1}{n} \sum_{i=1}^{n} c_i(w_t) = \nabla c(w)$.

How do we pick between 2nd order GD (Newton's method), 1st order GD, SGD? In terms of computations, cost = # of iterations $\times$ compute per iteration. The convergence criterion is

$$\|\nabla c(w_t)\|_2^2 \leq \varepsilon$$

for $\varepsilon > 0$.

For now, assume $\|\nabla c(w)\| \leq L$ for any w.

- GD with fixed stepsize $\eta > 0$: After t updates/iterations:

  $$\|\nabla c(w_t)\|_2^2 \leq \frac{2L}{t} (\underbrace{c(w_0) - c(w^*)}_{\text{how far start was from optimal}})$$

  So, $t \geq 2L/\varepsilon \cdot (c(w_0) - c(w^*))$ or $t = O(1/\varepsilon)$. For $\varepsilon = 0.1$, then $t = 1/\varepsilon = 10$. For $\varepsilon = 0.01$, then $t = 100$. For $t = O(1/\varepsilon)$, this is sublinear convergence. Progress slows down.

  GD takes $O(np/\varepsilon)$ compute to converge.

- 2nd-Order GD: $t = O(\log(1/\varepsilon))$. For $\varepsilon = 0.1$, $t = 1$. For $\varepsilon = 0.01$, $t = 2$.

  2nd-order GD takes $O(np^2 \log(1/\varepsilon))$.

  When is $np^2 \log(1/\varepsilon) < np/\varepsilon \implies p^2 \log(1/\varepsilon) < p/\varepsilon$, i.e. when is 2nd-order GD better than GD? For $p = 10, \varepsilon = 0.01$, $20 < 100$. For $p = 100, \varepsilon = 0.01$, $200 > 100$. So when we have a small number of weights/p, 2nd-order is faster.

- SGD vs. GD: SGD uses minibatches $\mathcal{B}$ of size b. If constant $\eta > 0$, almost the same convergence as GD. The convergence is

  $$O\left(\frac{1}{t}\right) + \frac{L \sigma_{\mathcal{B}}^2 \eta}{2}$$

  $\sigma_{\mathcal{B}}^2$ is related to noisiness of gradient.

  If $\eta_t = \eta/\sqrt{t}$, then $O(1/\sqrt{t})$, the oscillations eventually decay.

How can we keep $O(1/t)$ SGD convergence rate, but still reduce error term to 0? You can start with small minibatches, and increase them over time.

3 Generalized Linear Models

For classification, we had a Bernoulli distribution, $y \in \{0, 1\}$, and we learned $p(y = 1|x) = \sigma(\phi w)$. For linear regression, we had a Gaussian distribution, $y \in \mathbb{R}$, and we learned $p(y|x) = \mathcal{N}(\phi w, \sigma^2)$, where $\phi w = E[Y|x]$.

Others include Poisson distribution or multinomial distribution. $p(y|x)$ is an exponential family distribution.

3.1 Poisson Example

Imagine $y \in \{0, 1, \dots, \}$ such as number of accidents in a factory, conditioned on factory info x. The pmf for Poisson is

$$p(z) = \frac{\lambda^z \exp(-z)}{z!}$$

for $\lambda > 0$ is mean of Poisson distribution.

$$p(y|x) = \frac{\lambda(x)^y \exp(-y)}{y!}$$

Our goal is to learn $\lambda(x)$.

First attempt: $\lambda(x) = \phi w$, i.e. linear regression. This does not work since $\phi w < 0$, but $\lambda > 0$.

Second attempt: $\lambda(x) = \exp(\phi w)$. Getting closer.

3.2 General Form for GLM

Definition 3.1: General Form of Generalized Linear Models

$$p(y|x) = \exp(\phi w y - a(\phi w) + b(y))$$

where transfer function $g(\phi w) = a'(\phi w)$ and $E[Y|x] = g(\phi w)$.

E.g. Gaussian $a(\phi w) = \frac{1}{2}(\phi w)^2$, $g(\phi w) = a'(\phi w) = \phi w$.

$a(\theta) = \frac{1}{2}\theta^2, a'(\theta) = \theta, b(y) = -\frac{1}{2}y^2 + \ln \frac{1}{\sqrt{2\pi}}$. The general form is

$$\begin{aligned} p(y|x) &= \exp\left(\phi w y - \frac{1}{2}(\phi w)^2 - \frac{1}{2}y^2 + \ln \frac{1}{\sqrt{2\pi}}\right) \\ &= \exp\left(-\frac{1}{2}(\phi w - y)^2\right) \exp\left(\ln \frac{1}{\sqrt{2\pi}}\right) \\ &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(\phi w - y)^2\right) \end{aligned}$$

E.g. Bernoulli $a'(\theta) = \sigma(\theta), g(\phi w) = \sigma(\phi w)$.

E.g. Poisson $a(\theta) = \exp(\theta), a'(\theta) = \exp(\theta) = g(\theta), b(y) = -\ln y!$. The general form is

$$\begin{aligned} p(y|x) &= \exp((\phi w)y - \exp(\phi w) - \ln y!) \\ &= \exp(\phi w y)\exp(-\exp(\phi w))\exp(-\ln y!) \\ &= \frac{\exp(\phi w)^y \exp(-\exp(\phi w))}{y!} && (e^{ab} = (e^a)^b) \\ &= \frac{\lambda(x)^y \exp(-\lambda(x))}{y!} && (\lambda(x) = g(\phi w) = \exp(\phi w) \end{aligned}$$

3.3 GLM Update

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, $\phi_i = \phi(x_i)$,

$$c_i(w) = -\ln p(y_i|x_i, w) = -\ln \exp(\phi_i w y - a(\phi_i w) + b(y_i)) = -(\phi_i w)y_i + a(\phi_i w) + \text{constant}$$

The gradient is

$$\nabla c_i(w) = -y_i\phi_i^T + a'(\phi_i w)\phi_i^T = (a'(\phi_i w) - y_i)\phi_i^T = (\underbrace{g(\phi_i w)}_{\text{prediction}} - y_i)\phi_i^T$$

For linear regression, $(\phi_i w - y_i)\phi_i^T$, for logistic regression, $(\sigma(\phi_i w) - y_i)\phi_i^T$, and for Poisson regression, $(\exp(\phi_i w) - y_i)\phi_i^T$.

Recap on the three GLMs:

- **Linear regression**: For $y \in \mathbb{R}$

  $$g(\phi w) = \phi w \approx E[Y|x]$$

  Predict with $g(\phi w)$.

- **Logistic regression**: For $y \in \{0, 1\}$

  $$g(\phi w) = \sigma(\phi w) \approx E[Y|x]$$

  Predict $\mathbf{1}(\sigma(\phi w) > 0.5) \approx p(y = 1|x) = E[Y|x]$.

- **Poisson regression**: For $y \in \{0, 1, 2, \dots\}$

  $$g(\phi w) = \exp(\phi w) \approx E[Y|x]$$

  Predict $g(\phi w)$ or mode of Poisson$(\lambda = g(\phi w))$.

3.4 Multinomial Logistic Regression

This is another GLM and is multiclass classification. Given an input $x \in \mathbb{R}^{1\times d}$ and output $y \in \mathbb{R}^{1\times m} \in \{e_1^T, e_2^T, \dots, e_m^T\}$ where e_i^T is the row elementary vector with 1 in the ith coordinate for each class. For matrices, we have $X \in \mathbb{R}^{n\times d}$ and $Y = (y_1, \dots, y_n)^T \in \mathbb{R}^{n\times m}$.

$$p(y = [y_1, \dots, y_m]) = p(y_1 = 1)^{y_1} p(y_2 = 1)^{y_2} \cdots p(y_m = 1)^{y_m} = \alpha_1^{y_1}\alpha_2^{y_2}\cdots\alpha_m^{y_m}$$

where $\sum_{k=1}^{m} \alpha_k = 1$ and $\alpha_k \geq 0$.

E.g. $p(y = [0, 1, 0, \ldots, 0]) = \alpha_1^0 \alpha_2^1 \alpha_3^0 \cdots \alpha_m^0 = 0 = \alpha_2$.

In general, we want the probability given x,

$$p(y = [y_1, \ldots, y_m]) = \alpha_1(x)^{y_1} \cdots \alpha_m(x)^{y_m}$$

We cannot have a transfer function $\alpha_k(x) = \sigma(\phi w_k)$ as the probabilities may not sum to 1. We can normalize so the sum is 1, this leads to the softmax function.

The transfer function g is

$$\text{softmax}(\phi W) = \left[\underbrace{\frac{\exp(\phi w_1)}{\sum_{k=1}^{m} \exp(\phi w_k)}}_{\alpha_1(x)} \quad \cdots \quad \underbrace{\frac{\exp(\phi w_m)}{\sum_{k=1}^{m} \exp(\phi w_k)}}_{\alpha_m(x)} \right]$$

where $\phi \in \mathbb{R}^{1 \times p}$ and $W \in \mathbb{R}^{p \times m}$.

For each x_i,

$$p_i = \text{softmax}(\phi_i W) \in \mathbb{R}^{1 \times m}$$

So, $p_{i1} = p(y_i = 1 | x_i)$, $p_{i2} = p(y_i = 2 | x_i)$, and so forth.

E.g. $m = 4$, $p_i = [0.1, 0.4, 0.3, 0.2]$, $y_i = [0, 0, 1, 0]$. Select the class

$$\hat{y}_i = \underset{k \in \{1, \ldots, m\}}{\arg\max} \, \text{softmax}(\phi_i W) = [0, 1, 0, 0] \text{ or } 2$$

even though actual y_i is 3.

Why do we get $p_i = [0.1, 0.4, 0.3, 0.2]$, rather than $p_i = [0, 0.99, 0, 0.01]$?

- The true $p(y|x_i) = p_i$ (uncertainty, irreducible error).
- Could be insufficient data or optimization (uncertainty estimation).

GLM Update

$$c_i(W) = -\ln p(y_i | x_i), \frac{\partial c_i(W)}{\partial W_{jk}} = (p_{ik} - y_{ik}) \phi_{ij}$$

For all features,

$$\frac{\partial c_i(W)}{\partial W_{:k}} = (p_{ik} - y_{ik}) \phi_i^T$$

For the entire matrix,

$$\frac{\partial c_i(W)}{\partial W} = \left[\phi_i^T (p_{i1} - y_{i1}), \ldots, \phi_i^T (p_{im} - y_{im}) \right] = \phi_i^T (p_i - y_i) = g_t \in \mathbb{R}^{p \times m}$$

The update is

$$W_{t+1} = W_t - \eta_t \circ g_t$$

where $\circ$ is element-wise product.

An extra detail: we only need to learn $w_1, \ldots, w_{m-1} \in \mathbb{R}^{p \times 1}$ since we can infer $p(y_m = 1|x) = 1 - \sum_{k=1}^{m-1} p(y_k = 1|x)$. We can pivot with $w_m = 0$ and only update $w_1, \ldots, w_{m-1}$. This is w_m extra degrees of freedom.

Connection to Logistic Regression: For $m = 2$, $y_{LR} = 1 \implies y = [1, 0]$ and $y_{LR} = 0 \implies y = [0, 1]$. $w_1 = w, w_2 = 0$. Thus, we learn one weight vector $w_1 = w$ and pivot over w_2.

$$p(y_{LR} = 1|x) = \frac{\exp(\phi w_1)}{\exp(\phi w_1) + \exp(\phi w_2)} = \frac{\exp(\phi w)}{\exp(\phi w) + 1} = \sigma(\phi w) = \frac{1}{1 + \exp(-\phi w)}$$

We can learn w_2 with

$$p(y_{LR} = 0|x) = p(y = [0, 1]|x) = 1 - \sigma(\phi w)$$

3.5 Add Regularization to GLM

ℓ_2-Regularization:

$$\frac{1}{n} \sum_{i=1}^{n} c_i(w) + \frac{\lambda}{2} ||w||_2^2$$

where the first summand is the MLE part ($-\ln p(y|x)$) and the second summand is the Gaussian prior, so the entire thing is MAP.

Update: $w_{t+1} = w_t - \eta_t \circ \phi_i^T (p_i - y_i) - \lambda \eta_t \circ w_t$.

ℓ_1-Regularization:

$$\frac{1}{n} \sum_{i=1}^{n} c_i(w) + \lambda ||w||_1$$

where $||w||_1 = \sum_{j=1}^{p} |w_j|$. ℓ_1 is a V-shape function, so it has a stronger preference to reach $w = 0$. ℓ_2 has a strong preference for small w.

4 Constrained Optimization With Proximal Methods

4.1 Proximal Gradient Descent

We are given the optimization problem

$$\min_{w \in \mathbb{R}^p} c(w) + r(w)$$

where c is smooth (differentiable everywhere) and r is non-smooth.

E.g. ℓ_1-regularizer $r(w) = \lambda ||w||_1$. This is not differentiable at $w = 0$. For gradient descent, it will keep jumping back and forth between both sides, since the gradients do not get smaller.

E.g. Constraints $w \in [-1, 1]^p$,

$$r_{box}(w) = \begin{cases} 0 & \text{if } w \in [-1, 1]^p \\ \infty & \text{otherwise} \end{cases}$$

Recall gradient descent: Local Taylor series approximation

$$c(w) \approx \hat{c}_t(w) = c(w_t) + \nabla c(w_t)^T (w - w_t) + \frac{1}{2\eta_t} \|w - w_t\|_2^2$$

Then,

$$w_{t+1} = \arg\min_{w \in \mathbb{R}^p} \hat{c}_t(w), \nabla \hat{c}_t(w) = 0 \implies w_{t+1} = w_t - \eta_t \nabla c(w_t)$$

Now, we have a non-smooth $r(w)$, so

$$w_{t+1} = \arg\min_{w \in \mathbb{R}^p} \hat{c}_t(w) + r(w)$$

Using algebraic manipulation, we can get a equivalent optimization problem:

$$w_{t+1} = \arg\min_{w \in \mathbb{R}^p} \frac{1}{2} \left\| w - \underbrace{(w_t - \eta_t \nabla c(w_t))}_{\text{usual GD outcome}} \right\|_2^2 + \eta_t r(w)$$

Algorithm: Proximal Gradient Descent

1. $\tilde{w}_{t+1} = w_t - \eta_t \nabla c(w_t)$.
2. $w_{t+1} = \text{prox}_{\eta_t r}(w) = \arg\min_{w \in \mathbb{R}^p} \frac{1}{2} \|w - \tilde{w}_{t+1}\|_2^2 + \eta_t r(w)$ (proximal operator).

4.1.1 Box Constraints

For

$$r_{box}(w_j) = \begin{cases} 0 & w_j \in [a, b] \\ \infty \end{cases}$$

we have the proximal operator

$$\text{prox}_{\eta_t r_{box}}(v) = \begin{cases} a & \text{if } v < a \\ v & \text{if } v \in [a, b] \\ b & \text{if } v > b \end{cases}$$

Derivation: Minimization is separable.

$$\frac{1}{2} \sum_{i=1}^{p} (w_j - \tilde{w}_{t+1,j})^2 + \eta_t \sum_{j=1}^{p} r_{box}(w_j)$$

Independently,

$$\min_{w_j \in \mathbb{R}} \frac{1}{2} (w_j - \tilde{w}_{t+1,j})^2 + \eta_t r_{box}(w_j)$$

The solution w is the following:

- If $\tilde{w}_{t+1,j} \in [a, b]$, then $w_j = \tilde{w}_{t+1,j}$.
- If $\tilde{w}_{t+1,j} < a$, then $w_j = a$.
- If $\tilde{w}_{t+1,j} > b$, then $w_j = b$.

4.1.2 ℓ_1-Regularization

For $r(w) = \lambda||w||_1 = \lambda\ell_1(w)$,

$$\text{prox}_{\eta_t\lambda\ell_1}(\tilde{w}_{t+1,j}) = \begin{cases} \tilde{w}_{t+1,j} - \eta_t\lambda & \text{if } \tilde{w}_{t+1,j} > \eta_t\lambda \\ 0 & \text{if } |\tilde{w}_{t+1,j}| < \eta_t\lambda \\ \tilde{w}_{t+1,j} + \eta_t\lambda & \text{if } \tilde{w}_{t+1,j} < -\eta_t\lambda \end{cases}$$

4.1.3 Linear Regression

The first step is $\tilde{w}_{t+1} = w_t - \eta_t(\phi_i w_t - y_i)\phi_i^T$. The second step is

$$w_{t+1} = \text{prox}_{\eta_t\lambda\ell_1}(\tilde{w}_{t+1})$$

element-wise.

5 Representation Learning

6 Latent Factors

Identify key features (factors) $h \in \mathbb{R}^{1\times p}$ for an observation $x \in \mathbb{R}^{1\times d}$ so that

$$x \approx hD$$

for dictionary $D \in \mathbb{R}^{p\times d}$. This is finding a linear weighting that approximately produces the observed inputs.

We let $D_k \in \mathbb{R}^{1\times d}$ be a representative instance that maximally has h_k.

E.g. Let $x \in \mathbb{R}^{30}$ be patient info and we $p = 5$ number of traits for the patients. We would have a dictionary $D \in \mathbb{R}^{5\times 30}$.

Think of h as being $\phi(x)$ from before. How do we learn h and dictionary D?

6.1 Matrix Factorization

Principal component analysis (PCA): Suppose we want $p < d$ features, i.e. dimensionality reduction. For example, if $X \in \mathbb{R}^{n\times d}$, then we want to break it up into $X = HD$, where $H \in \mathbb{R}^{n\times p}$ and $D \in \mathbb{R}^{p\times d}$. To do this, we solve the optimization problem

$$\min_{D\in\mathbb{R}^{p\times d}} \sum_{i=1}^{n} \|x_i - h_i D\|_2^2$$

for all h_i, where $\hat{x}_i = h_i D$ and $\hat{X} \approx X$ for $\hat{X} = HD$.

Using the SVD of $X = U\Sigma V^T$, we have $\hat{X} = U_p \Sigma_p V_p^T$. So,

$$X - \hat{X} = \sum_{j=1}^{d} \sigma_j u_j v_j^T - \sum_{j=1}^{p} \sigma_j u_j v_j^T = \sum_{j=p+1}^{d} \sigma_j u_j v_j^T$$

for smallest $\sigma_{p+1} \geq \cdots \geq \sigma_d$.

For the PCA solution, we have $H = U_p \Sigma_p$ (magnitudes), $D = V_p^T$ (unit length), and $\hat{X} = HD$.

For new data x_{new},

$$h_{new} = \arg\min_{h \in \mathbb{R}^p} \|x_{new} - h_{new} D\|_2^2 = x_{new} D^T$$

Sparse coding: For $p > d$, we want a small subset of dictionary to recreate x.

$$\min_{D \in \mathbb{R}^{p \times d}} \frac{1}{n} \sum_{i=1}^{n} \|x_i - h_i D\|_2^2 + \lambda ||h_i||_1 + \lambda \sum_{j=1}^{p} \|D_j\|_2^2$$

ℓ_1 makes many elements of h_i zero. This is why it is called sparse coding. The last ℓ_2 makes sure that D is not high in magnitude if ℓ_1 drives h_i down to 0.

We need ℓ_1 otherwise we can have the solution $H = [X, 0]$ and $D = [I, 0]^T$ and $HD = X$, but this is a nonsense solution.

For x_{new},

$$h_{new} = \arg\min_{h \in \mathbb{R}^p} \|x_{new} - hD\|_2^2 + \lambda ||h||_1$$

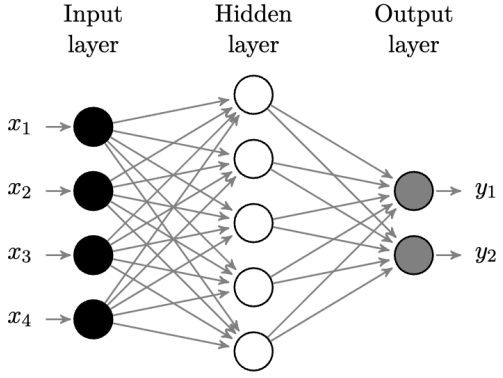
6.2 Probabilistic Approach

We can do probabilistic PCA.

$$p(x|hD) = \mathcal{N}(\mu = hD, \sigma^2 I)$$

7 Neural Networks

We will use neural networks to learn representations. So suppose we have an input x and we want



to learn the representation h. Each element of h can be learned as a linear weighting with x.

$$h_1 = f(xW_{:1}), h_2 = f(xW_{:2}), \ldots, h_k = f(xW_{:k}) \implies h = f(xW)$$

where f is the activation function. Some examples of activation functions are

$$f(\theta) = \sigma(\theta) = \frac{1}{1+\exp(-\theta)}, f(\theta) = \tanh(\theta) \in [-1,1], f(\theta) = \text{ReLU}(\theta) = \max(\theta, 0)$$

What h does this operation produce? First, xW acts like a filter and tells you how similar you are to something. The activation function, such as ReLU, can then partition the space, i.e. tell you what part is active.

7.1 Fully-Connected Feedforward Neural Networks

Suppose we have a fully-connected k-layer network. We label $W^{(1)}$ from the end of network and the corresponding layer $h^{(1)}$ all the way back to the first layer $W^{(k)}$ and layer h_k.

The last layer $\hat{y} = f^{(1)}(h^{(1)}W^{(1)})$, where $f^{(1)}$ is based on properties of y (GLM choice). Each layer is like a GLM on $h^{(i)}$.

Activation function choice example: If $y \in \mathbb{R}$, then $f^{(1)} = \text{identity}$. If $y \in \{1,2,3\}$, $f^{(1)} = \text{softmax}$.

Each layer will have an activation function, i.e.

$$h^{(i)} = f^{(i+1)}(xW^{(i+1)})$$

The neural network is then

$$\hat{y} = f_w(x)$$

7.2 Learning f_W using Backpropagation

The backpropagation algorithm is simply gradient descent on the loss for a neural network, with a careful ordering of computation to avoid repeating computation.

1. Pick objective $y \in \mathbb{R}$, $c_i(W) = \frac{1}{2}(f_W(x_i) - y_i)^2$.
   For $y \in \{0,1\}$, $c_i(W) = y_i \ln f_W(x_i) - (1-y_i)\ln(1 - f_W(x_i))$ and $f^{(1)} = \text{sigmoid}$.
2. Use gradient descent: need to compute $\nabla c_i(W)$.

$$W_t = (W_t^{(1)}, W_t^{(2)}, \ldots, W_t^{(k)}) \implies W_{t+1} = W_t - \eta_t \nabla c_i(W_t)$$

Two Layer example: We have input x, $W^{(2)}$ to layer h_1, $W^{(1)}$ to output.

For the output transition, $\theta^{(1)} = h^{(1)}W^{(1)}$. Let the loss be $\ell(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$.

$$\frac{\partial c_i(W)}{\partial W^{(1)}} = \frac{\partial \ell(\hat{y}_i, y_i)}{\partial W^{(1)}} = \underbrace{\frac{\partial \ell(\hat{y}_i, y_i)}{\partial \theta^{(1)}}}_{\delta \text{ for GLMs}} \frac{\partial \theta^{(1)}}{\partial W^{(1)}} = (f^{(1)}(\theta^{(1)}) - y_i)h_1 = \delta^{(1)}h^{(1)}$$

$$\begin{aligned}
\frac{\partial c_i(W)}{\partial W_{jk}^{(2)}} &= \frac{\partial \ell(\hat{y}_i, y_i)}{\partial \theta^{(1)}} \frac{\partial \theta^{(1)}}{\partial W_{jk}^{(2)}} \\
&= \frac{\partial \ell(\hat{y}_i, y_i)}{\partial \theta^{(1)}} \frac{\sum_{i=1}^{p_1} \partial h_i^{(1)} W_i^{(1)}}{\partial W_{jk}^{(2)}} \\
&= \frac{\partial \ell(\hat{y}_i, y_i)}{\partial \theta^{(1)}} (0 + 0 + \cdots + \frac{\partial h_k^{(1)} W_k^{(1)}}{\partial W_{jk}^{(2)}} + 0 + \cdots + 0) \\
&= \frac{\partial \ell(\hat{y}_i, y_i)}{\partial \theta^{(1)}} \cdot W_k \frac{\partial h_k^{(1)}}{\partial W_{jk}^{(2)}} \\
&= \frac{\partial \ell(\hat{y}_i, y_i)}{\partial \theta^{(1)}} \cdot W_k \cdot \frac{\partial f^{(2)}(\theta_k^{(2)})}{\partial \theta_k^{(2)}} \frac{\partial \theta_k^{(2)}}{\partial W_{jk}^{(2)}} \\
&= \frac{\partial \ell(\hat{y}_i, y_i)}{\partial \theta^{(1)}} \cdot W_k \cdot f'^{(2)}(\theta_k^{(2)}) x_{ij}
\end{aligned}$$

where $\theta_k^{(2)} = x W_{:k}^{(2)}$.

E.g. $f^{(2)} = \sigma$, $\sigma'(\theta) = (1 - \sigma(\theta))\sigma(\theta)$.
$f^{(2)} = \text{ReLU}$,

$$f'^{(2)} = \begin{cases} 1 & \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

Putting it all together,

$$\delta_k^{(2)} = \delta^{(1)} W_k^{(1)} f'^{(2)}(\theta_k^{(2)})$$
$$\frac{\partial c_i(W)}{\partial W_{jk}^{(2)}} = \delta_k^{(2)} x_{ij}, \frac{\partial c_i(W)}{\partial W^{(1)}} = \delta_k^{(1)} h^{(1)}$$

The error for $\delta_j^{(\ell+1)}$ is

$$\delta_j^{(\ell+1)} = \left(\sum_k W_{jk}^{(\ell)} \delta_k^{(\ell)} \right) f^{(\ell+1)}(\theta_j^{(\ell+1)})$$

Some questions:

- What if there are two outputs? Before, $\delta_k^{(2)} = W_k^{(1)} \delta^{(1)} f'(\theta_k^{(2)})$, now

$$\delta_k^{(2)} = (W_{k1}^{(1)} \delta_1^{(1)} + W_{k2}^{(1)} \delta_2^{(1)}) f'(\theta_k^{(2)})$$

- What is $\delta_1^{(1)}$ and $\delta_2^{(2)}$?
  For $y \in \mathbb{R}$, $f^{(1)}$ is the identity.

$$\delta_1^{(1)} = (h^{(1)} W_{:1}^{(1)} - y_{i1}), \delta_2^{(1)} = (h^{(1)} W_{:2}^{(1)} - y_{i2})$$

  For $y \in \{0, 1, 2, \dots\}$, then Poisson regression which uses $f^{(1)} = \exp$, so

$$\delta_1^{(1)} = \exp(h^{(1)} W_{:1}^{(1)}) - y_{i1}, \delta_2^{(1)} = \exp(h^{(1)} W_{:2}^{(1)}) - y_{i2}$$

- Is loss per output always separately added up?

  No, multinomial logistic regression.

All the above derivation is to compute the gradient $g_t = \nabla c_i(W_t)$. Now we can use g_t in any of the optimizers like SGD with fixed stepsize, Adam update, or proximal update if we have a non-smooth r.

A nuance of neural networks is that initialization of W_0 matters a lot (this is the starting point for gradient descent). In GLMs, we could initialize $w_0 = 0$, the convex function would converge to w^*. For example, if $W^{(2)} = 0$, then $h = \max(xW^{(2)}, 0) = 0$ for ReLU activation. $\delta^{(1)} = (hW^{(1)} - y)h = 0$ and $\delta_j^{(2)} = W_j^{(1)}\delta^{(1)} f'(\theta_j) = 0$. There is no update and so it is stuck.

For an ideal choice, we obviously would like to choose near optimal weights, but this is not possible. Some heuristics would be randomly initialize W where we keep variances uniform between layers.

7.3 Other Network Architectures

Definition 7.1: Sparse Neural Networks

Less weights than a fully-connected neural network.

Advantages: computational, biologically-inspired bias.

Definition 7.2: Skip Connection

Connecting nodes across layers.

This is the avoid vanishing gradients.

8 Regularization and Auxiliary Losses on Neural Networks

8.1 Autoencoders

Definition 8.1: Autoencoder

A neural network that has the same solution as PCA.

A linear autoencoder is for $x \in \mathbb{R}^d$ with $p < d$ and $W = (W^{(1)}, W^{(2)})$, we have a neural network to a hidden layer of p dimension to $\hat{x}$.

$$c_i(W) = \frac{1}{2}\left\| \underbrace{x_i W^{(2)} W^{(1)}}_{\hat{x}_i} - x_i \right\|_2^2$$

The bottleneck: $W^{(2)}W^{(1)} = \tilde{W}$ with $\text{rank}(\tilde{W}) \leq p < d$.

Using SVD,

$$\tilde{W} = U_d \Sigma_d V^T = U_d \, diag(\sigma_1, \sigma_2, \ldots, \sigma_p, 0, \ldots, 0) V^T$$

For PCA:

$$\min_D \sum_{i=1}^n \|x_i - h_i D\|_2^2$$

For Autoencoders:

$$\min_{W^{(2)},W^{(1)}} \sum_{i=1}^n \left\|x_i W^{(2)} W^{(1)} - x_i\right\|_2^2$$

Recall the solution for PCA,

$$H = U_p \Sigma_p, D = V_p^T$$

The solution for autoencoders is

$$W^{(1)} = V_p^T, W^{(2)} = V_p$$

Check: $XW^{(2)}W^{(1)} = U_d \Sigma_d V^T V_p V_p^T = U_d \Sigma_d \begin{bmatrix} I_p \\ 0 \end{bmatrix} V_p^T = U_p \Sigma_p V_p^T = \hat{X}$.

Using PCA, $\min_h \|x_{new} - hD\|$. Using autoencoders, $h_{new} = x_{new} W^{(2)}$.

If you run backprop autoencoders, will you get $W^{(2)} = V_p, W^{(1)} = V_p^T$? No, $W^{(2)} Q Q^{-1} W^{(1)}$ where Q is any $p \times p$ invertible matrix and now we have a different solution.

8.2 Supervised Autoencoder

Jointly optimize representation h that maintains most information X, but also useful for predicting y. So the neural network will predict $\hat{x}$ as well as y.

$$\lambda \min_{W^{(2)},W^{(1)}} \sum_{i=1}^n \left\|x_i W^{(2)} W^{(1)} - x_i\right\|_2^2 + \sum_{i=1}^n \ell(x_i W^{(2)} W^{(1)} y, y_i)$$

where ℓ is any GLM loss.

An alternative is to

1. Do autoencoder to get $W^{(2)}, W^{(1)}$.
2. Learn $f_{W_y^{(1)}} \approx E[Y|x]$

The loss from $\hat{x}$ is called auxiliary, since it acts like a regularization. where $W^{(1)}$ is a $p \times (d+1)$ matrix now.