Deep Learning

Instructor: Tongyao Pang

Notes Taker: Zejin Lin

TSINGHUA UNIVERSITY.

linzj23@mails.tsinghua.edu.cn

lzjmaths.github.io

March 7, 2025

Contents

1 Regression

$$\min_{\omega \in \mathbb{R}^m} \frac{1}{2N} \|\Phi\omega - y\|^2 + \lambda C(\omega) \tag{1.1}$$

Lasso: $C = \|\omega\|_1$. **Ridge regression**: $C = \|\omega\|_2$.

subgradient of f:

$$\partial f(x_0) = \{g | f(x) \geqslant f(x_0) + g^T(x - x_0)\}$$

In particular,

$$\partial|x| = \begin{cases} 1, & x > 0 \\ -1, & x < 0 \\ [-1, 1], & x = 0 \end{cases}$$

1.1 Binary classification problem

one-hot encoding for the output $\{\binom{1}{0}, \binom{0}{1}\}$. It can be understood as the probability for each class and can take continuous values.

A **linear hypothesis space** is $\{u(x) : u = \omega^T x, x \in \mathbb{R}^n, \omega \in \mathbb{R}^n\}$.

Softmax:Map the extracted feature u to the space of one-hot codes

$$\mu = \frac{1}{1 + e^{-u}}, \quad 1 - \mu = \frac{e^{-u}}{1 + e^{-u}} = \frac{1}{1 + e^{u}}$$

$$KL(p, q) = \int p(\log p - \log q) \tag{1.2}$$

For p real probability, to minimize (1.2), suffices to minimize

$$-\int p \log q_\theta \mathrm{d}x = -\sum_{x_i} \log q_\theta(x_i)$$

which is called **Maximum likelihood (cross entropy)**

$$-\sum \log p(y_i | x_i, \omega) = \sum -y_i \log \mu_i - (1 - y_i) \log(1 - \mu_i)$$

We reduce to minimize the thing above.

1.2 Gradient Descent

$$J(\theta) = \sum_{i=1}^{N} L(f_\theta(x_i), y_i), \quad \theta^{t+1} = \theta^t - \eta_t \frac{\partial J(\theta)}{\partial \theta}|_{\theta=\theta^t}$$

For **empirical loss**

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} J_i(\theta), J_i(\theta) L(f_\theta(y_i), x_i)$$

Stochastic Gradient Descent:only compute gradients over a mini batch for each epoch

$$\theta_{k+1} = \theta_k - \eta \frac{1}{B} \sum_{i \in I_B} \nabla J_i(\theta_k)$$

I_B, called **mini batch** are randomly sampled from the training data indexes.

The motivation to sample is that for distribution x, $\dfrac{x_1 + \cdots + x_N}{N}$ has the same mean μ but less variance $\frac{1}{N}\sigma^2$, so in order to have more randomness and greatly reduce the computational cost, we choose less ammount of data.

Randomness can help avoid getting stuck in local minimum.

SGD: $x_{t+1} = x_t - \alpha \nabla f(x_t)$.

SGD+**momentum**:

$$v_{t+1} = \rho v_t + \nabla f(X_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

v_t is the momentum which helps accelerate convergence by accumulating the gradients of past steps and smoothing out the oscillations, where $\rho = 0.9$ or 0.99.

1.3 Adaptive learning rate

$$r_t = r_{t-1} + \nabla f(x_t) \bigodot \nabla f(x_t)$$

$$x_{t+1} = x_t - \frac{\alpha}{\sqrt{r_t + \varepsilon}} \bigodot \nabla f(x_t)$$

For frequent features, the updates will be smaller, and for rare features, the updates will be larger.

Notation $\odot$ is the multiplication for each component, which means:

$$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \odot \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_1 b_1 \\ \vdots \\ a_n b_n \end{pmatrix}$$

Exponential Moving Averaging(EMA)

$$r_t = \beta r_{t-1} + (1-\beta)\nabla f(x_t) \odot \nabla f(x_t)$$

$$x_{t+1} = x_t - \frac{\alpha}{\sqrt{r_t + \varepsilon}} \bigodot \nabla f(x_t)$$

RMSProp uses a moving average, avoiding overly aggresive decay complared with AdaGrad.

Adaptive Moment Estimation(Adam):RMAProp+Momentum

$$g_t = \nabla f(x_t)$$

$$v_t = \beta_1^t v_{t-1} + (1-\beta_1^t) g_t, r_t = \beta_2^t r_{t-1} + (1-\beta_2^t) g_t \odot g_t$$

$$v_t = \frac{v_t}{1-\beta_1^t}, E_t = \frac{E_t}{1-\beta_2^t}$$

$$x_{t+1} = x_t - \frac{\alpha}{\sqrt{r_t + \varepsilon}} \odot v_t$$

2 Vanishing or Exploding Gradient

The gradient of the Sigmoid function is very small most of the time, leading to vanishing gradients

We want to avoid exploding or vanishment in gradient.

Sigmoid: $\sigma(z) = \dfrac{1}{1+e^{-z}}$.

Tanh: $\sigma(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$.

ReLU: $\mathrm{ReLU}(z) = \begin{cases} z, & z > 0 \\ 0, & \text{otherwise} \end{cases}$.

LeakyReLU: $\mathrm{LeakyReLU}(z) = \begin{cases} z, & z > 0 \\ az, & \text{otherwise} \end{cases}$. The gradient neither vanishes nor explodes; it is computationally fast, but some neurons may not be activated.

LeakyReLU solves the issue with ReLU and is the most commonly used.

Consider the back propagation $\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial J}{\partial x}W$. Cumulate after multi-layers

$$\mathrm{Var}(\frac{\partial J}{\partial x}) = \prod_i n_l \mathrm{Var}(W_l)\mathrm{Var}(\frac{\partial J}{\partial x_l})$$

We want $n_l \mathrm{Val}(W_l) \sim 1$.

After normalizing their variances, the updating becomes more steady and efficient.

To avoid variance becoming too small or too large in deep layers, normalize features in the network

$$\widehat{x}_i = \frac{x_i - \mathbb{E}x_i}{\sqrt{\mathrm{var}(x_i)}}$$

Batch Normalization: normalize features across samples within each batch.

Index

List of Theorems