Deep Learning

David Zhao

Febrary 9, 2023

Preface

This paper is a learning documentaion which follows the book Dive into Deep learning.

Preliminaries

Linear Algebra

Linear Regression

We assume that the relationship between features $\vec{x}$ and target y is approximately linear, i.e,

$$E[Y|X = \vec{x}] = x_1w_1 + \ldots + x_dw_d + b \tag{1}$$

where d is the *feature dimensionality*, and b is the *bias*. As such,

$$\hat{y} = \vec{w}^T\vec{x} + b = X\vec{w} + b \tag{2}$$

In essence, our goal is to find parameters $\vec{w}$ and b such that our prediction error is minimized for new data examples that are sampled from the same distribution X.

Loss Function

Naturally, our model requires an objective measure of how well it fits the training data. Loss functions play this role by quantifying the distance between the *observed* and *predicted* labels. The most commonly used loss function is the squared error.

$$l^{(i)}(\vec{w}, b) = \frac{1}{2}(\hat{y}^{(i)} - \vec{y}^{(i)})^2 \tag{3}$$

Note that the constant coefficient $\frac{1}{2}$ is only notationally convenient as it disappears when we take the derivative of the loss function. Furthermore, large differences between estimates $\hat{y}^{(i)}$ and targets $\vec{y}^{(i)}$ lead to larger contributions due to the function's quadratic form. In fact, while it does encourage our model to avoid sizeable errors, it also yields an excessive sensitivity to anomalous data. Finally, to evaluate our model's performance over entire dataset of n examples, we simply take the average of the losses on the training set:

$$L^{(i)}(\vec{w}, b) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(\hat{y}^{(i)} - \vec{y}^{(i)})^2 \tag{4}$$

Clearly, our goal is to find parameters $\vec{w}^*$ and b^* to minimize the total loss across all examples.

Minibatch Stochastic Gradient Descent

$$(\vec{w}, b) \leftarrow (\vec{w}, b) - \frac{\eta}{|\beta|} \sum_{i \in \beta_i} \frac{\partial}{\partial(\vec{w}, b)} l^{(i)}(\vec{w}, b)$$

$$\vec{w} \leftarrow \vec{w} - \frac{\eta}{|\beta|} \sum_{i \in \beta_i} \frac{\partial}{\partial \vec{w}} l^{(i)}(\vec{w}, b) = \vec{w} - \frac{\eta}{|\beta|} \sum_{i \in \beta_i} \vec{x}^{(i)}(\vec{w}^T \vec{x}^{(i)} + b - \vec{y}^{(i)})$$

$$b \leftarrow b - \frac{\eta}{|\beta|} \sum_{i \in \beta_i} \frac{\partial}{\partial b} l^{(i)}(\vec{w}, b) = \vec{w} - \frac{\eta}{|\beta|} \sum_{i \in \beta_i} (\vec{w}^T \vec{x}^{(i)} + b - \vec{y}^{(i)})$$

Normal Distribution and Squared Loss

Recall

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

Assume observations arise from noisy measurements

$$y = \vec{w}^T \vec{x} + b + \epsilon$$

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - \vec{w}^T \vec{x} - b)^2\right)$$

$$P(y|X) = \prod_{i=1}^{n} P(\vec{y}^{(i)}|\vec{x}^{(i)})$$

since all pairs $(\vec{y}^{(i)}, \vec{x}^{(i)})$ were drawn independently. But, maximizing the product of exponential functions is tedious. Instead, we minimize the negative log-likelihood:

$$\begin{aligned} -\log(y|X) &= -log(\prod_{i=1}^{n} P(\vec{y}^{(i)}|\vec{x}^{(i)})) \\ &= \sum_{i=1}^{n} \frac{1}{2}\log\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}(\vec{y}^{(i)} - \vec{w}^T \vec{x}^{(i)} - b)^2 \end{aligned}$$

As such, it follows that minimizing the square error loss is equivalent to the maximum likelihood estimation of a linear model under additive Gaussian noise.

Generalization

The phenomenon of our model fitting closer to the training model than to the underlying distribution is called *overfitting*. Instead, our goal is to train our model in such a way that it may find a generalizable pattern and make correct predictions about previously unseen data.

Training Error & Generalization Error

In standard supervised learning setting, we assume the training and testing data to be drawn independently from identical distributions (i.e. *IID* assumption). Training error (R_{emp}) is a statistic calculated on the training dataset:

$$R_{emp}[X, \vec{y}, f] = \frac{1}{n} \sum_{i=1}^{n} l(\vec{x}^{(i)}, \vec{y}^{(i)}, f(\vec{x})^{(i)})$$

Generalization error (R) is an expectation taken with respect to the underlying distribution:

$$R[p, f] = E_{(\vec{x},y)\sim P[l(\vec{x},y,f\vec{x})]} = \int\int l(\vec{x}, y, f(\vec{x}))p(\vec{x}, y)d\vec{x}dy$$

Note that we can never measure R exactly since the density function $p(\vec{x}, y)$ has a form that cannot be precisely known. Moreover, since we cannot sample an infinite stream of data points, we must resort to estimating the generalization error by applying our model to an independent test set that is withheld from our training set.

Model Complexity

Intuitively, when we have simple models mixed with abundant data, the training and generalization error tend to be close. Conversely, we can expect more a complex model and/or fewer examples to cause our training error to diminish, but the generalization error to grow. Error on the holdout data, i.e. the validation set, is called the validation error.

Polynomial Curve Fitting

Cross Validation

Weight Decay

Recall that we can always mitigate overfitting by collecting more training data. However, gathering more data is often costly, time consuming, etc. Therefore, we introduce our first *regularization* technique known as *weight decay*.

Note that we may also limit model complexity by tweaking the degree of our fitted polynomial. However, even small changes in degree can dramatically increase model complexity, hence motivating the use of weight decay.

Norms & Weight Decay