Deep Learning

David Zhao

Febrary 9, 2023

1 Preface

This paper is a learning documentation adapted/copied from the book Dive into Deep learning. Coding implementations are omitted (for now).

2 Preliminaries

AutoDifferentiation

Linear Algebra

Norms

Some of the most useful operators in linear algebra are norms. Informally, the norm of a vector tells us how big it is. Here, we are employing a notion of size that concerns the magnitude of a vector's components (not its dimensionality).

A norm is a function $||\cdot||$ that maps a vector to a scalar and satisfies the following three properties:

1. Given any vector $\mathbf{x}$, if we scale the vector by a scalar $\alpha \in \mathbb{R}$, its norm scales accordingly:
$$||\alpha\mathbf{x}|| = |\alpha|||\mathbf{x}||$$
2. For any vectors $\mathbf{x}$ and $\mathbf{y}$, norms must satisfy the triangle inequality:
$$||\mathbf{x} + \mathbf{y}|| = ||\mathbf{x}|| + ||\mathbf{y}||$$
3. The norm of a vector is nonnegative and vanishes if and only if the vector is zero:
$$||\mathbf{x}|| > 0 \iff \mathbf{x} \neq 0$$

For instance, the $l2$ norm measures the (Euclidean) length of a vector which we've all seen already in school when calculating the hypotenuse of a right triangle.. Formally, we know it as

$$||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{n} \mathbf{x}_i^2}$$

The $l1$ norm is also popular and the associated metric is called the Manhattan distance. By definition, the norm sums the absolute values of a vector's elements:

$$||\mathbf{x}||_1 = \sum_{i=1}^{n} |\mathbf{x}_i|$$

Both the $l1$ and $l2$ norms are special cases of the more general norms:

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} \mathbf{x}_i^p \right)^{1/p} \tag{1}$$

Chain Rule

Let $y = f(\mathbf{u})$ such that $\forall i, \mathbf{u}_i = g_i(\mathbf{x})$ where $\mathbf{u} = (u_1, u_2, \ldots, u_m)$ and $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Then

$$\frac{\partial y}{\partial x_i} = \frac{\partial y}{\partial u_1} \frac{\partial u_1}{\partial x_i} + \ldots + \frac{\partial y}{\partial u_m} \frac{\partial u_m}{\partial x_i} \tag{2}$$

Baye's Theorem

Given any event A and B such that $P(B) \neq 0$,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3}$$

Expectations

3 Linear Neural Networks for Regression

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

$$E[Y|X = \mathbf{x}] = x_1 w_1 + \ldots + x_d w_d + b \tag{4}$$

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

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b \quad = \mathbf{X}\mathbf{w} + b \tag{5}$$

In essence, our goal is to find parameters $\mathbf{w}$ 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 or unwell it fits the training data. Loss functions fill in this role by quantifying the distance between the *observed* and *predicted* labels. The most commonly used loss function is the squared error.

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}(\hat{y}^{(i)} - \mathbf{y}^{(i)})^2 \tag{6}$$

Note that the presence of the constant coefficient $\frac{1}{2}$ is notationally convenient as it disappears when we take the derivative of the loss function. Also notice that large differences

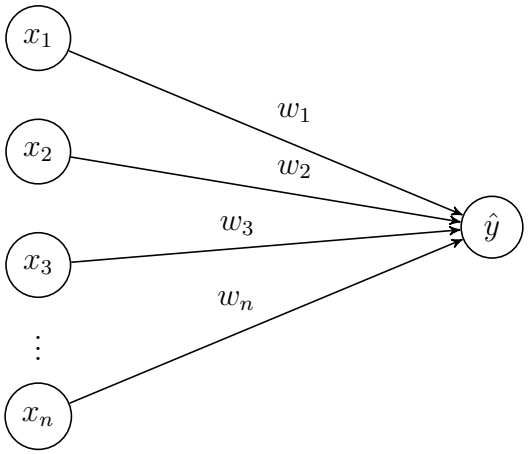

Figure 1: Linear Regression Model with feature dimensionality n

between estimates $\hat{y}^{(i)}$ and targets $\mathbf{y}^{(i)}$ 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 the dataset of n examples, we simply take the average of the losses on the training set:

$$L^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(\hat{y}^{(i)} - \mathbf{y}^{(i)})^2 \tag{7}$$

Hence, our goal is to find parameters $\mathbf{w}^*$ and b^* such that the total loss is minimized across all examples.

Minibatch Stochastic Gradient Descent

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

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

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

Normal Distribution and Squared Loss

So far we have given a fairly functional motivation of the squared loss objective: the optimal parameters return the conditional expectation whenever the underlying pattern is truly linear, and the loss assigns outsize penalties for outliers. We can also provide a more formal motivation for the squared loss objective by making probabilistic assumptions about the distribution of noise.

To begin, recall that a normal distribution with mean μ and variance σ^2 (standard deviation σ) is given as

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

One way to motivate linear regression with squared loss is to assume that observations arise from noisy measurements, where the noise is normally distributed as follows:

$$y = \mathbf{w}^T\mathbf{x} + b + \epsilon \quad \text{where } \epsilon \sim \mathcal{N}(0,\, \sigma^2)$$

Thus, we can now write out the likelihood of seeing a particular y for a given $\mathbf{x}$ via

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

As such, the likelihood factorizes. According to the *principle of maximum likelihood*, the best values of parameters $\mathbf{w}$ and b are those that maximize the likelihood of the entire dataset:

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

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

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

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:

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

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

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

Note that we can never measure R exactly since the density function $p(\mathbf{x}, y)$ has a form that can almost never 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

In cases when we are dealt with scarce training data, it is likely that we often lack enough hold out data to form a validation set. A popular solution is to use *K-fold cross-validation* where the training data is first partitioned into k disjoints sets. Then, we perform a total of k training/validation steps, each time training on $k-1$ sets and validating on the remaining unused set. Finally, we average the training and validation errors over the results obtained from our k experiments.

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 our necessity for a more fine-tuning method, i.e. weight decay.

Norms & Weight Decay

$$\mathbf{w} \leftarrow (1-\eta\lambda)\mathbf{w} - \frac{\eta}{|\beta|}\sum_{i\in\beta_i}\frac{\partial}{\partial \mathbf{w}} l^{(i)}(\mathbf{w}, b) = \mathbf{w} - \frac{\eta}{|\beta|}\sum_{i\in\beta_i}\mathbf{x}^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b - \mathbf{y}^{(i)})$$

4 Linear Neural Networks for Classification

Softmax Regression

$$= \hat{y}_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)}$$

5 Multilayer Perceptrons

Recall in section 3, we described affine transformations as linear transformations with added bias. This model maps inputs directly to outputs via a single affine transformation, followed by a softmax operation. However, linearity is often a strong assumption.

Limitations of Linear Models

Linearity implies the weaker law of *monotonicity* i.e. any increase in inputs must always correspond to an increase in our model's output (positive weights), or a decrease in our model's output (negative weights). Often times, linearity becomes too strong of an assumption to be applied to problems that require more specific modelling. Suppose for example we want to predict whether an individual will repay loan based on their salary. Although this relationship is monotonic, it is perhaps not linear as an increase in income from \$0 to \$50,000 likely corresponds to a higher likelihood of repayment than an increase from \$1 million to \$1.05 million. As such, it may be preferable to post-process our outcome by using a logarithmic map, to make linearity a more plausible assumption.

Incorporating Hidden Layers

We overcome the limitations of linearity by incorporating one or more hidden layers. The most common way to do this is to stack many fully connected layers on top of each other. We can think of the first $L-1$ layers as our representation and the final layer as our linear predictor. This architecture is commonly called a *multilayer preceptron* or *MLP*.

From Linear to Nonlinear

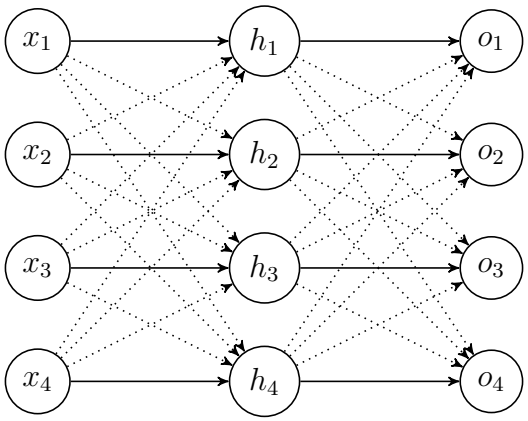

Figure 2: An MLP with a hidden layer of 4 hidden units.

Activation Functions

ReLU Function

The most popular choice, due to both simplicity of implementation and its good performance on a variety of predictive tasks, is the rectified linear unit (ReLU) ReLU provides a very simple nonlinear transformation. Given an element x, the function is defined as the maximum of that element and 0:

$$ReLU(x) = \max(x, 0)$$

When the input is negative, the derivative of the ReLU function is 0, and when the input is positive, the derivative of the ReLU function is 1. Note that the ReLU function is not differentiable when the input takes value precisely equal to 0. In these cases, we default to the left-hand-side derivative and say that the derivative is 0 when the input is 0.

Sigmoid Function

$$sigmoid(x) = \frac{1}{1 + \exp(-x)}$$

Tanh Function

$$tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

Variational autoencoders Standard distribution non-standard distribution gaussian What is the difference between why do we initialize parameters with gaussian noise How are activation fuctions used