Loss Functions

CHEN Si

Contents

1 Introduction

1. In most cases, our parametric model defines a distribution $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$ and we simply use the principle of maximum likelihood.
    - This means we use the **cross-entropy** between the training data and the model's predictions as the cost function.
2. Sometimes, we merely predict some statistic of $\boldsymbol{y}$ conditioned on $\boldsymbol{x}$
    - Specialized loss functions enable us to train a predictor of these estimates.
3. The total loss function used to train a neural network often combine a primary loss function with a regularization term.

1.1 Learning Conditional Distribution with Maximum Likelihood

Most modern neural networks are trained using maximum likelihood.

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x})$$

- This means that the cost function is simply the negative log-likelihood, equivalently described as the cross-entropy between the training data and the model distribution.

1.1.1 Advantages

1. Deriving the loss function from maximum likelihood removes the burden of designing cost functions for each model.

2. Helps to avoid the problem of saturation

   - Several output units involve an exp function that can saturate when its argument is very negative, because of the log function in the negative log-likelihood loss function.
   - Prevent the gradient vanishing caused by saturation.

1.1.2 Example

If $p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; f(\boldsymbol{x};\boldsymbol{\theta}), \boldsymbol{I})$, then we recover the mean squared error cost

$$J(\boldsymbol{\theta}) = \frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x};\boldsymbol{\theta})\|^2 + \text{const}$$

1.2 Learning Conditional Statistics

- Instead of learning a full probability distribution $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$, we often want to learn just one conditional statistic of $\boldsymbol{y}$ given $\boldsymbol{x}$. (e.g. The mean of $\boldsymbol{y}$ given $\boldsymbol{x}$.)

- The loss function can be viewed as a mapping from functions (rather than a set of parameters) to real numbers.

1.2.1 Two results derived from calculus of variations

Different cost functions give different statistics.

1. Solving the optimization problem

$$f^* = \arg\min_f \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x})\|^2$$

   yields

$$f^*(\boldsymbol{x}) = \mathbb{E}_{\mathbf{y}\sim p_{\text{data}}(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}]$$

   so long as this function lies within the class we optimize over.

   - If we could train on infinitely many samples from the true data generating distribution, minimizing the MSE cost function would give a function that predicts the **mean** of $\boldsymbol{y}$ for each value of $\boldsymbol{x}$.

2. Solving the optimization problem

$$f^* = \arg\min_f \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x})\|_1$$

   yields a function that predicts the **median** value of $\boldsymbol{y}$ for each $\boldsymbol{x}$, as long as such a function may be described by the family of functions we optimize over.

   - This cost function is commonly called **mean absolute error**.

Disadvantages

- MSE and MAE often lead to poor results when used with gradient-based optimization. Some output units that saturate produce very small gradients when combined with these cost functions.
  - This is one reason that the cross-entropy cost function (negative log-likelihood) is more popular than MSE or MAE, even when it is not necessary to estimate an entire distribution $p(\boldsymbol{y} \mid \boldsymbol{x})$.

2 Output Units

The choice of how to represent the ouput then determines the form of the cross-entropy function.
Throughout this section, we suppose that the feedforward network provides a set of hidden features defined by $\boldsymbol{h} = f(\boldsymbol{x}; \boldsymbol{\theta})$.

2.1 Linear Units

Definition

$$\hat{\boldsymbol{y}} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$$

- Linear output layers are often used to produce the mean of a conditional Gaussian distribution

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$$

Advantages

1. Because linear units do not saturate, they pose little difficulty for gradient-based optimization algorithms.

2.2 Sigmoid Units for Bernoulli Output Distributions

Definition

$$\hat{y} = \sigma\left(\boldsymbol{w}^\top \boldsymbol{h} + b\right)$$

- $z = \boldsymbol{w}^\top \boldsymbol{h} + b$ defining such a distribution over binary variables is called a **logit**.

- Predicts only $P(y = 1 \mid \boldsymbol{x})$
  (Actually predicts unnormalized log probability $\log \tilde{P}(y = 1 \mid \boldsymbol{x}) = z$)

Motivation Construct an unnormalized probability distribution $\tilde{P}(y)$. $\forall y = 0, 1$, if we assume

$$\log \tilde{P}(y) = yz$$

After exponentiating we obtain

$$\tilde{P}(y) = \exp(yz)$$

After normalizing we obtain

$$\begin{aligned} P(y) &= \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(y'z)} \\ &= \sigma((2y-1)z) \end{aligned}$$

Advantages

1. **Ensures there is always a strong gradient whenever the model has the wrong answer.**

2. Prevent the saturation of Sigmoid if we use MLE.

$$\begin{aligned}J(\boldsymbol{\theta}) &= -\log P(y \mid \boldsymbol{x}) \\ &= -\log \sigma((2y-1)z) \\ &= \zeta((1-2y)z)\end{aligned}$$

   Thus, gradient-based learning can act to quickly correct a mistaken z, because the softplus function asymptotes toward simply returning its argument $(1-2y)z = |z|$ if z have the wrong sign.

3. MLE is almost always the preferred approach to training sigmoid output units.

   - If we use MSE, the gradient can shrink too small whether the model has the correct answer or thr incorrect answer.

In Software Implementation

1. Write the negative log-likelihood as a function of z, rather than as a function of $\hat{y} = \sigma(z)$

   - To avoid numerical underflow of Sigmoid, which yields numerical overflow of logarithm.

Applications

1. Binary classifier.

2.3 Softmax Units for Multinoulli Output Distributions

Definition First, predicts unnormalized log probabilities

$$\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$$

where

$$z_i = \log \tilde{P}(y = i \mid \boldsymbol{x})$$

The softmax function can then exponentiate and normalize $\boldsymbol{z}$ to obtain the desired $\hat{\boldsymbol{y}}$

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- $\forall i \in 1, 2, \ldots, n,\ \mathrm{softmax}(\boldsymbol{z})_i \in (0, 1)$
- $\displaystyle\sum_i \mathrm{softmax}(\boldsymbol{z})_i = 1$

Properties

1. Invariant to adding the same scalar to all its inputs

$$\mathrm{softmax}(\boldsymbol{z}) = \mathrm{softmax}(\boldsymbol{z} + c)$$

2. A "softened" (continuous and differentiable) version of the $\arg\max$ (with its result represented as a one-hot vector).

3. The corresponding soft version of the maximum function is $\mathrm{softmax}(\boldsymbol{z})^\top \boldsymbol{z}$.

Advantages

1. As with the logistic sigmoid, the use of the exp function works well when training the softmax to output a target value y using maximum log-likelihood.

2. The log in the log-likelihood undoes the exp of the softmax

$$\log \mathrm{softmax}(\boldsymbol{z})_i = z_i - \log \sum_j \exp(z_j)$$

   - The fist term shows that the input z_i always has a direct contribution to the cost funtion because this term cannot saturate.
   - $\log \sum_j \exp(z_j) \approx \max_j z_j$
     - Always penalizes the most active incorrect prediction.
     - If the correct answer already has the largest input to the softmax, then the two terms will roughly cancel.
   - When maximizing the log-likelihood, the first term encourages z_i to be pushed up, while the second term encourages all of $\boldsymbol{z}$ to be pushed down.

3. Unregularized maximum likelihood will drive the model to learn parameters that drive the softmax to predict the fraction of counts of each outcome observed in the training set

$$\mathrm{softmax}(\boldsymbol{z}(\boldsymbol{x}; \boldsymbol{\theta}))_i \approx \frac{\sum_{j=1}^m \mathbf{1}_{y^{(j)}=i, \boldsymbol{x}^{(j)}=\boldsymbol{x}}}{\sum_{j=1}^m \mathbf{1}_{\boldsymbol{x}^{(j)}=\boldsymbol{x}}}$$

- Because MLE is a consistent extimator, this if guaranteed to happen as long as the model family is capable of representing the training distribution.

4. Squared error (or other loss function with out log) is a poor loss function for softmax units and can fail to train the model to change its output, even when the model makes highly confident incorrect predictions.

    - The softmax activation can saturate when the differences between input values become extreme.
        - An output $\text{softmax}(\boldsymbol{z})_i$ saturates to 1 when the corresponding input is maximal ($z_i = \max_i z_i$) and z_i is much greater than all the other inputs.
        - An output $\text{softmax}(\boldsymbol{z})_i$ saturates to 0 when the corresponding input z_i is not maximal and the maximum is much greater.
    - The gradient will vanish when the argument to the exp becomes very negative.

Applications

1. Used as the output of a multiclass-classifier, to represent the probability distribution over n different classes.

2. More rarely, used inside the model itself, if we wish the model to choose between one of n different options for some internal variable.

2.4 Other Output Types

(To be accomplished) (See *Deep Learning Book 6.2.2.4*)