Loss Functions

CS 4277 Deep Learning

Kennesaw State University

Loss Functions

The goal of a (eager) machine learning algorithm is to find the parameters of a model that produces the best possible mapping from inputs to outputs.

- We do this using a training data set $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$
- Training uses feedback from the mismatch betweeen the model's predicted $\hat{y}$s and the ground truth ys.
- A *loss function* returns a single number that represents this mismatch.
- So finding the best possible mapping from inputs to outputs reeduces to minimizing the loss function.

Density Estimation

Say we have N observations of a scalar x which we denote with $\mathbf{x} = (x_1, \ldots, x_N)$.

- Estimating the distribution given the data is known as *density estimation*.
- We must assume a distribution, so we're estimating the parameters of the distribution.
- We assume data points are drawn independently and are identically-distributed.
  - This is known as the i.i.d. assumption

Example: Likelihood of the Gaussian

Since our data set is i.i.d., the probability of the data set is

$$p(\mathbf{x}|\mu, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(x_n|\mu, \sigma^2)$$

This is known as the *likelihood function* for the Gaussian.

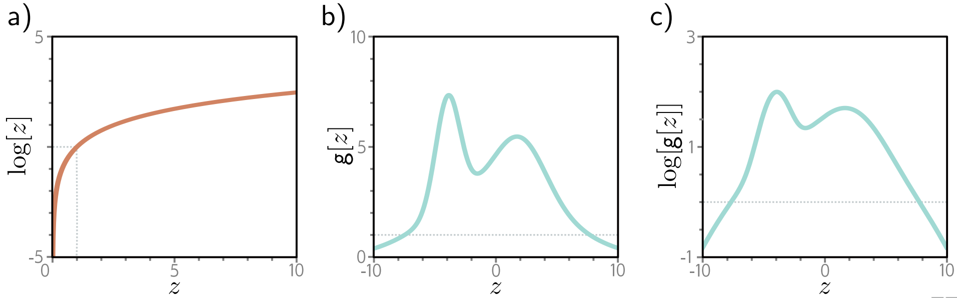
Maximum Likelihood

Finding the parameters of a distribution that maximize the probability of the observed data is known as *maximum likelihood estimation* (MLE).

In pracice, we transform likelihood functions into log likelihood functions.

Why log:

- Log of a function monotonically increasing and concave – $\operatorname{argmax} \ln(f) = \operatorname{argmax} f$
- Log easy to work with: $\ln(ab) = \ln(a) + \ln(b)$, $\ln(\frac{a}{b}) = \ln(a) - \ln(b)$, $\ln e^x = x$
- Multiplying probabilities can underflow – summing logs avoids this problem



Log Likelihood of Gaussian

For the Gaussian likelihood function we saw earlier, the log likelihood

$$p(\mathbf{x}|\mu,\sigma^2) = \prod_{n=1}^{N} \mathcal{N}(x_n|\mu,\sigma^2)$$

$$p(\mathbf{x}|\mu,\sigma^2) = \prod_{n=1}^{N} \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp(-\frac{1}{2\sigma^2}(x_n-\mu)^2)$$

$$\ln p(\mathbf{x}|\mu,\sigma^2) = \sum_{n=1}^{N} \ln(\frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp(-\frac{1}{2\sigma^2}(x_n-\mu)^2))$$

$$\ln p(\mathbf{x}|\mu,\sigma^2) = \sum_{n=1}^{N} \ln(1) - \ln(\sqrt{2\pi}) - \ln(\sigma) - \frac{(x_i-\mu)^2}{2\sigma^2}$$

Maximum Likelihood of Gaussian

If we take the partial derivative of the Gaussian log likelihood function with respect to μ, set it to zero, and solve for μ we get:

$$\mu_{ML} = \frac{1}{N} \sum_{n=1}^{N} x_n$$

If we take the partial derivative with respect to σ^2 we get:

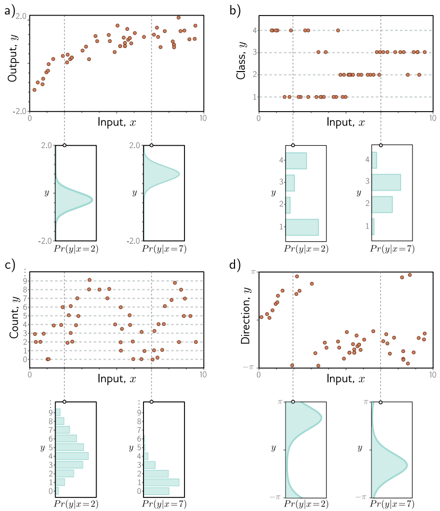
$$\sigma^2_{ML} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu_{ML})^2$$

These should look familiar. They are the sample mean and sample variance of the Gaussian.

Loss Functions and Machine Learning Models

We seek a model $\boldsymbol{f}(\boldsymbol{x}, (\phi))$ that computes a $\hat{\boldsymbol{y}}$ given an $\boldsymbol{x}$.

- We can recast this problem as the computation of a conditional probability $p(\boldsymbol{y}_i\ \boldsymbol{x}_i)$.
- Minimizing the loss corresponds to maximizing this conditional probability.



General Maximum Likelihood Criterion

We choose a parametric distribution defined over the output domain $\boldsymbol{y}$ then train our model to compute the paramters, $\boldsymbol{\theta}$ of this distribution.

- If we choose a Gaussian distribution, then $\boldsymbol{\theta} = \{\mu, \sigma^2\}$.

We want to find the parameters of the model $\hat{\boldsymbol{\phi}}$ that maximize the conditional probability distribution $P(\boldsymbol{y}_i|\boldsymbol{\theta}_i)$ for all $\boldsymbol{y}_i$s.

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left(\prod_{i=1}^{I} p(\boldsymbol{y}_i|\boldsymbol{x}_i)\right)$$

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left(\prod_{i=1}^{I} p(\boldsymbol{y}_i|\boldsymbol{\theta}_i)\right)$$

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left(\prod_{i=1}^{I} p(\boldsymbol{y}_i|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\phi}))\right)$$

KENNESAW STATE UNIVERSITY

Maximizing Log-Likelihood

Recalling that the total likelihood of the training data is:

$$P(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_I | \boldsymbol{x}_1, \ldots, \boldsymbol{x}_I) = \prod_{i=1}^{I} p(\boldsymbol{y}_i | \boldsymbol{x}_i)$$

which is impractical due to underflow, so we prefer to maximize the log-likelihood:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}}(\prod_{i=1}^{I} p(\boldsymbol{y}_i | \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\phi}))$$

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}}(\log(\prod_{i=1}^{I} p(\boldsymbol{y}_i | \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\phi}))$$

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}}(\sum_{i=1}^{I} log(p(\boldsymbol{y}_i | \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\phi}))$$

KENNESAW STATE UNIVERSITY

Minimizing Negative Log-Likelihood

By convention we minimize the loss function. We can turn a maximization problem into a minimization problem by multiplying by -1.

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left(-\sum_{i=1}^{I} log(p(\boldsymbol{y}_i|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\phi}))\right)$$
$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}(L(\boldsymbol{\phi}))$$

Which is the final form of our loss function $L(\boldsymbol{\phi})$

Inference

Our network now computes a probability distribution over $\boldsymbol{y}$ instead of predicting $\hat{y}$. To get a prediction, we return the maximum of the distribution:

$$\hat{\boldsymbol{y}} = \underset{y}{\mathrm{argmax}}(p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\phi}))$$

This is often computed in terms of the parameters $\boldsymbol{\theta}$ predicted by the model. E.g., for Gaussian the maximum is at μ

Recipe for Constructing and Using Loss Functions

Now that we understand MLE for loss functions, we can create a recipe for constructing loss functions for training data $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{I}$ using the maximum likelihood approach:

1. Choose a suitable probability distribution $P(\boldsymbol{y}|\boldsymbol{\theta})$ defined over the predictions (output domain) $\boldsymbol{y}$ with distributoin parameters $\boldsymbol{\theta}$.
2. Set the machine learning model $\boldsymbol{f}(\boldsymbol{x}, \phi)$ to predict one or more of these parameters, so $\boldsymbol{\theta} = \boldsymbol{f}(\boldsymbol{x}, \phi)$ and $P(\boldsymbol{y}|\boldsymbol{\theta}) = P(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\phi}))$.
3. To train the model, find the network parameters $\hat{\phi}$ that minimize the negative log-likelihood loss function over the training data pairs $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}(L(\boldsymbol{\phi})) = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}(-\sum_{i=1}^{I} log(p(\boldsymbol{y}_i|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\phi}))$$

4. To perform inference for a new test example $\boldsymbol{x}$, return either the full distribution $P(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\phi}))$ or the value where the distribution is maximized.

Now we see this recipen applied to three common problems: univariate regression, binary classification, and multiclass classification.

Example 1: Univariate Regression

Goal: predict a single output $y \in \mathbb{R}$ from input $\boldsymbol{x}$ using model $f(\boldsymbol{x}, \boldsymbol{\phi})$.

1. Choose a distribution over output y.

$$p(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma * 2}} \exp(-\frac{(y-\mu)^2}{2\sigma^2})$$

2. Set the machine learning model $f(\boldsymbol{x}, \boldsymbol{\phi})$ to compute one or more parameters of the distribution. Here, we choose only the mean $\mu = f(\boldsymbol{x}, \boldsymbol{\phi})$:

$$p(y|f(\boldsymbol{x}, \boldsymbol{\phi}), \sigma^2) = \frac{1}{\sqrt{2\pi\sigma * 2}} \exp(-\frac{(y-f(\boldsymbol{x}, \boldsymbol{\phi}))^2}{2\sigma^2})$$

3. Calculate the negative log-likelihood for our chosen parameter(s) to use as the loss function. We already did this for the Gaussian earlier; here we substitute $f(\boldsymbol{x}, \boldsymbol{\phi})$ for μ and y_i for x_i since we're calculating the distribution parameter over the output:

$$L(\boldsymbol{\phi}) = \sum_{i=1}^{I} \ln(1) - \ln(\sqrt{2\pi}) - \ln(\sigma) - \frac{(y_i - f(\boldsymbol{x}, \boldsymbol{\phi}))^2}{2\sigma^2}$$

3.1: Least-Squares Loss Function

Our negative log-likelihood function is unwieldy, so do some algebraic manipulations.

$$L(\boldsymbol{\phi}) = \sum_{i=1}^{I} \ln(1) - \ln(\sqrt{2\pi}) - \ln(\sigma) - \frac{(y_i - f(\boldsymbol{x}, \boldsymbol{\phi}))^2}{2\sigma^2}$$

First, remove terms that don't depend on ϕ:

$$L(\boldsymbol{\phi}) = \sum_{i=1}^{I} \frac{(y_i - f(\boldsymbol{x}, \boldsymbol{\phi}))^2}{2\sigma^2}$$

Next, remove denominator since it's a positive scaling factor that doesn't affect the position of the minimum:

$$L(\boldsymbol{\phi}) = \sum_{i=1}^{I} (y_i - f(\boldsymbol{x}, \boldsymbol{\phi}))^2$$

And this is the familiar least squares loss function you get when you assume that your outputs Y are i.i.d. and each $y_i \sim \mathcal{N}(y_i|, f(\boldsymbol{x}, \boldsymbol{\phi}), \sigma^2)$

4: Inference

The model doesn't predict y directly, instead it predicts the mean of the Gaussian distributoin assumed to generate y. We turn that into a predicted y with:

$$\hat{y} = \underset{y}{\operatorname{argmax}}(p(y|f(\boldsymbol{x}, \hat{\boldsymbol{\phi}}), \sigma^2))$$

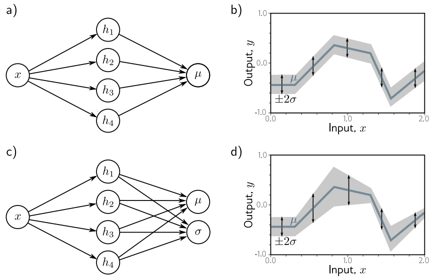
Since for the univariate Gaussian the mean μ is the most likely value, inference is simply

$$\hat{y} = f(\boldsymbol{x}, \hat{\boldsymbol{\phi}})$$

Heteroscedatic vs Homoscedatic Regression

- In *homoscedatic* regression, the variance is constant across all data points.
- In *heteroscedatic* regression, the variance varies as a function of the input data.

To handle heteroscedatic regression, you can train a network to to compute both mean and variance.



$$\mu = f_1(\boldsymbol{x}, \boldsymbol{\phi})$$

$$\sigma^2 = f_2(\boldsymbol{x}, \boldsymbol{\phi})^2 \qquad \text{(Square output to ensure positive)}$$

The loss function is then:

$$\hat{\boldsymbol{\phi}} = \underset{\phi}{\operatorname{argmin}} \left(-\sum_{i=1}^{I} \left(\log\left(\frac{1}{\sqrt{2\pi f_2(\boldsymbol{x}, \boldsymbol{\phi})^2}}\right) - \frac{(y_i - f_1(\boldsymbol{x}, \boldsymbol{\phi}))^2}{2 f_2(\boldsymbol{x}, \boldsymbol{\phi})^2} \right) \right)$$

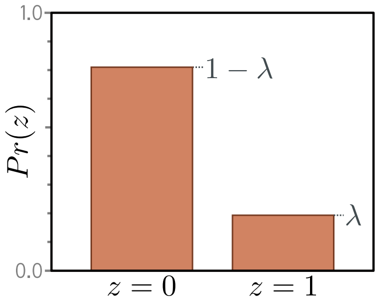
Example 2: Binary Classification

1. Choose Bernoulli distribution, which is defined over the output space $y \in \{0, 1\}$. The parameter λ represents the probability that $y = 1$.

$$p(y|\lambda) = \begin{cases} 1 - \lambda & y = 0, \\ \lambda & y = 1 \end{cases}$$

Or, equivalently:

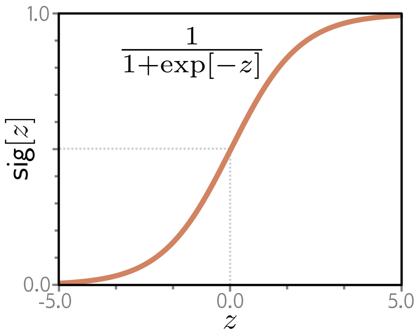
$$p(y|\lambda) = (1 - \lambda)^{1-y} \lambda^{y}$$



2: Predicting λ

2. Set the network $f(\boldsymbol{x}, \boldsymbol{\phi})$ to predict the single parameter λ. Since λ is a probability, but the network can return a number outside $[0, 1]$, we squash the output to the required range using the *logistic sigmoid* function:

$$sig(z) = \frac{1}{1 + \exp(-z)}$$



Binary Loss Function and Inference

3. Calculate the negative log-likelihood for our parameter, λ, to use as the loss function.

The likelihood function is:

$$p(y|\boldsymbol{x}) = (1 - sig(f(\boldsymbol{x}, \boldsymbol{\phi})))^{1-y} sig(f(\boldsymbol{x}, \boldsymbol{\phi}))^{y}$$

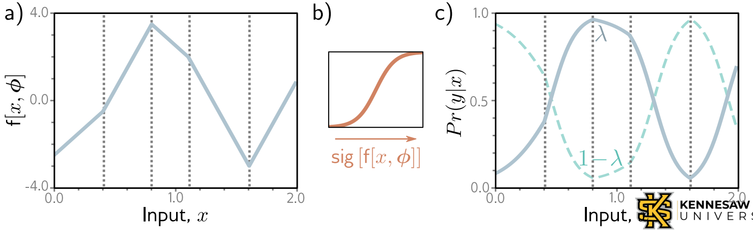
and the negative log-likelihood loss function is:

$$L(\boldsymbol{\phi}) = \sum_{i=1}^{I} (1 - y_i) \log\left(1 - sig(f(\boldsymbol{x}_i, \boldsymbol{\phi}))\right) - y_i \log\left(sig(f(\boldsymbol{x}_i, \boldsymbol{\phi}))\right)$$

This is known as *binary cross-entropy loss*. (More on that later . . .)

4. Inference is simply:

$$\hat{y} = \begin{cases} 1 & \text{if } \lambda > 0.5, \\ 0 & \text{otherwise} \end{cases}$$



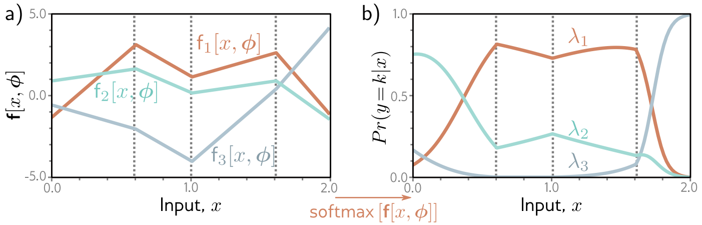
Example 3: Multiclass Classification

1. We choose a *categorical distribution* with K parameters $\lambda_1, \lambda_2, \ldots, \lambda_K$ representing the probability of each of the corresponding K categories $y \in \{1, 2, \ldots, K\}$.

$$p(y = k) = \lambda_k$$

2. We set the network $f(\boldsymbol{x}, \boldsymbol{\phi})$ to predict each of the K parameters for a given input $\boldsymbol{x}$. Since the parameters must sum to 1 to be a valid probability distribution and the network can produce arbitrary values, we pass each output through the *softmax* function:

$$softmax_k(z) = \frac{\exp(z_k)}{\sum_{k'=1}^{K} \exp(z_{k'})}$$



Multiclass Loss and Inference

3. Calculate the negative log-likelihood for our parameters, λ_k, to use as the loss function.

The likelihood function is:

$$p(y = k|\boldsymbol{x}) = softmax_k\left(f(\boldsymbol{x}, \boldsymbol{\phi})\right)$$

The negative log-likelihood loss function is:

$$\begin{aligned} L(\boldsymbol{\phi}) &= -\sum_{i=1}^{I} \log\left(softmax_{y_i}\left(f(\boldsymbol{x}_i, \boldsymbol{\phi})\right)\right) \\ &= -\sum_{i=1}^{I}\left(f_{y_i}(\boldsymbol{x}_i, \boldsymbol{\phi}) - \log\left(\sum_{k'=1}^{K} \exp(f_{k'}(\boldsymbol{x}_i, \boldsymbol{\phi}))\right)\right) \end{aligned}$$

where $f_{y_i}(\boldsymbol{x}_i, \boldsymbol{\phi})$ is the y_ith output and $f_{k'}(\boldsymbol{x}_i, \boldsymbol{\phi})$ is the k'th output of the network.

4. Inference is simply the most probable category:

$$\hat{y} = \underset{k}{\operatorname{argmax}}\left(p(y = k|\boldsymbol{f}(\boldsymbol{x}, \hat{\boldsymbol{\phi}}))\right)$$

KENNESAW STATE UNIVERSITY