Application of KL divergence, Jensen–Shannon divergence and Wasserstein Distance in Machine Learning

Shuangcheng Liu

521021910904

Abstract

KL divergence is an important concept in the information theory, and this paper will introduce the applications of KL divergence in deep learning as a loss function. Based on its mathematical properties, this paper will also introduce Jensen–Shannon divergence and Wasserstein Distance to compensate for the imperfection of KL divergence in some applications.

Keyword

KL divergence; loss function; Jensen–Shannon divergence; Wasserstein Distance

Contents

1 Applications of KL Divergence in Deep Learning as a Loss Function

In deep learning, we want the model to produce an output distribution that is similar to the ground truth distribution. KL divergence can be used to measure the difference between the model's output distribution and the ground truth distribution, which can help guide the model training.

1.1 Definition and Properties of KL Divergence

1.1.1 Definition

Given two discrete probability distributions P and Q, the KL divergence is defined as:

$$D_{KL}(P||Q) = \sum_{i=1}^{n} P(i) \log \frac{P(i)}{Q(i)}$$

where $P(i)$ and $Q(i)$ are the probabilities of the $i-th$ discrete value for the two distributions.

1.1.2 Non-negativity

$D_{KL}(P||Q) \geq 0$, and equality holds only when $P = Q$.

1.1.3 Asymmetry

$D_{KL}(P||Q) \neq D_{KL}(Q||P)$.

1.1.4 Not satisfying the triangle inequality

$D_{KL}(P||R) \nleq D_{KL}(P||Q) + D_{KL}(Q||R)$.

1.2 Relation with Maximum likelihood

Let $P_{data}(X, y)$ denote the ground truth distribution of all potential samples. Suppose we have n training samples generated from $P_{data}(X, y)$, then our discriminative model is trained on $(X_i, y_i) \sim P_{data}(X, y)$ with $i = 1, 2, ..., n$ by maximizing:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(y_i|X_i)$$

According to the Large number theory:

$$\frac{1}{n} \sum_{i=1}^{n} \log p_\theta(y_i|X_i) \approx \sum_{X \in \Omega} P_{data}(X) \log p_\theta(y|X) = E_{P_{data}}[\log p_\theta(y|X)]$$

So Maximizing log-likelihood is equivalent to :

$$\begin{aligned} \max_\theta \mathcal{L}(\theta) &= \max_\theta \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(y_i|X_i) \\ &= \min_\theta \{-E_{P_{data}}[\log p_\theta(y|X)]\} \\ &= \min_\theta E_{P_{data}}[\log P_{data}(y|X) - E_{P_{data}}[\log p_\theta(y|X)]] \\ &= \min_\theta D_{KL}(P_{data}(y|X)||p_\theta(y|X)) \end{aligned}$$

where $P_{data}(y|X)$ denotes the ground truth conditional probability, so it certainly does not depend on the model θ, indicating that $E_{P_{data}}[\log P_{data}(y|X)]$ is a constant.

With $P_{data}(y|X)$ representing the perfect ideal training model, $p_\theta(y|X)$ denotes the model we have learned. According to the meaning of KL divergence, $D_{KL}(P_{data}(y|X)||p_\theta(y|X))$ denotes the distance between the model we have trained and the ground truth model from the God's perspective.

Likely we can conclude that for discriptive models, the essence of the maximum likelihood is to minimize the KL divergence between the model $p_\theta(X)$ and the ground truth $P_{data}(X)$.

2 Jensen–Shannon divergence, introduced from GAN

2.1 background

Suppose that we have a known prior distribution on h, with $h \sim p(h)$. The decoder defines a generative model, which is a non-linear generalization of the factor analysis model. This model can be learned by generative adversarial networks (GAN), where we pair the generator model G with a discriminator model D. D(X) denotes the probability that X is a true objective instead of a generated one. We can train the pair of (G, D) by an adversarial scheme. Let:

$$\begin{aligned} V(D,G) &= E_{P_{data}}[\log p_D(X)] + E_{h\sim p(h)}[\log(1-p_D(G(h)))] \\ &= E_{P_{data}}[\log D(X)] + E_{h\sim p(h)}[\log(1-D(G(h)))] \end{aligned}$$

for a given θ, let p_θ be the distribution of $g_\theta(h)$ where $h \sim p(h)$, We can find a perfect discriminator assuming equal numbers of real and fake samples:

$$\begin{aligned} &\frac{\partial V}{\partial D} = 0 \\ &\Rightarrow \frac{P_{data}(X)}{D(X)} - \frac{p_\theta(X)}{1-D(X)} = 0 \\ &\Rightarrow D(X) = \frac{P_{data}(X)}{P_{data}(X)+p_\theta(X)} \end{aligned}$$

2.2 Definition and properties of Jensen–Shannon divergence

2.2.1 Definition

Given two discrete probability distributions P and Q:

$$JSD(P||Q) = \frac{D_{KL}(P||p_{mix}) + D_{KL}(Q||p_{mix})}{2}$$

where $p_{mix} = \frac{P+Q}{2}$.

2.2.2 Non-negativity

$JSD(P||Q) \geq 0$, and equality holds only when $P = Q$.

2.2.3 symmetry

$JSD(P||Q) = JSD(Q||P)$.

2.2.4 Not satisfying the triangle inequality

$JSD(P||R) \nleq JSD(P||Q) + JSD(Q||R)$.

2.3 Relation between Jensen–Shannon divergence and GAN

We learn discriminator model D and generator model G by $\min_G \max_D V(D,G)$, now we want to prove that to minimize $V(D,G) \Leftrightarrow$ to minimize $JSD(P_{data}|p_\theta)$:

$$\begin{aligned}
2JSD(P_{data}||p_\theta) &= D_{KL}(p_\theta||p_{mix}) + D_{KL}(P_{data}||p_{mix}) \\
&= \sum_X [p_\theta(X)\log\frac{p_\theta(X)}{p_{mix}(X)} + P_{data}(X)\log\frac{P_{data}(X)}{p_{mix}(X)}] \\
&= -H(p_\theta) - H(P_{data}) - \sum_X [p_\theta(X)\log\frac{p_\theta(X)}{2(1-D(X))} + P_{data}(X)\log\frac{P_{data}(X)}{2D(X)}] \\
&= -H(p_\theta) - H(P_{data}) + H(p_\theta) + H(P_{data}) \\
&+ \sum_X [p_\theta(X)\log[2(1-D(X))] + P_{data}(X)\log 2D(X)] \\
&= \sum_X [(p_\theta(X) + P_{data}(X))\log 2] + \sum_X [p_\theta(X)\log[(1-D(X))] + P_{data}(X)\log D(X)] \\
&= 2\log 2 + V(D,G)
\end{aligned}$$

From the proof above we can find out the advantage of Jensen–Shannon divergence over KL Divergence in symmetry. KL-divergence measures the difference between two distributions, but it is sensitive to the order of the two distributions. $D_{KL}(P||Q) \neq D_{KL}(Q||P)$, which means that KL-divergence may lead to inconsistent optimization directions for the generator and discriminator.

On the other hand, Jensen–Shannon divergence measures the average similarity between two distributions, taking into account not only the distance between generated and real data, but also their similarity, so that it is easier for the generator and discriminator to jointly optimize in the same direction.

2.4 Introduction of Wasserstein Distance

2.4.1 Definition

$$W_p(P||Q) = \left(\inf_{\gamma \in \Pi(P,Q)} \int_{\mathbb{R}^d \times \mathbb{R}^d} |x - y|^p d\gamma(x, y) \right)^{1/p}$$

2.4.2 Non-negativity

$W(P||Q) \geq 0$, and equality holds only when $P = Q$.

2.4.3 symmetry

$W(P||Q) = W(Q||P)$.

2.4.4 satisfying the triangle inequality

$W(P||R) \leq W(P||Q) + W(Q||R)$

Since the Wasserstein distance satisfies the triangle inequality, it is a well defined distance metric that can more accurately measure the distance between two distributions. The advantage of the Wasserstein distance over KL divergence and Jensen–Shannon divergence is that even if the support sets of two distributions do not overlap or overlap very little, it can still reflect the distance between the two distributions. In contrast, Jensen–Shannon divergence is constant under this circumstance, and KL divergence may be meaningless.

3 My own concern on the topic

During my research about the topic, what always confuses me is that we can always improve the way we define the distance between distributions based on the mathematical properties we need, which then turns out to be a new way to train the model.

However, there are so many ways to improve the definition, Jensen–Shannon divergence is obviously not the only way to make KL divergence symmetric. So I am confused that how can we know that we are on the right path, and I do not regard learning accuracy as a convincing evidence.

A Reference

https://www.zhihu.com/tardis/zm/art/93853664?source$_id = 1005$
$https://blog.csdn.net/Zhaohui_Zhang/article/details/120546731$
$(2022-2023-2)-CS3308-02lecture1$
$(2022-2023-2)-CS3308-02lecture4$

B Testing code

```
import tensorflow as tf
from scipy.stats import wasserstein_distance, entropy
import numpy as np
import matplotlib.pyplot as plt

def generate_vectors(n_samples):
    return np.random.normal(size=[n_samples, 784])

def sample_z(m, n):
    return np.random.normal(size=[m, n])

def generate_images(generator, n_samples):
    z = sample_z(n_samples, 100)
    images = generator(z)
    images = (images + 1) / 2.0
    return images

def make_generator_model():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Dense(256, input_shape=(100,),
        use_bias=False))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.LeakyReLU(alpha=0.2))
    model.add(tf.keras.layers.Dense(512, use_bias=False))
```

```
        model.add(tf.keras.layers.BatchNormalization())
        model.add(tf.keras.layers.LeakyReLU(alpha=0.2))
        model.add(tf.keras.layers.Dense(784, activation='tanh',
            use_bias=False))
        return model

    generator = make_generator_model()
    generated_images = generate_images(generator, 10000)
    generated_distributions = tf.reshape(generated_images, [-1,
        784])
    random_vectors = generate_vectors(10000)

    w_distances =
        [wasserstein_distance(generated_distributions[i],
        random_vectors[i]) for i in range(10000)]

    entropies = [entropy(generated_distributions[i], base=2)
        for i in range(10000)]
    fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10,4))
    ax1.hist(w_distances, bins=50)
    ax1.set_xlabel("Wasserstein Distance")
    ax1.set_ylabel("Frequency")
    ax2.hist(entropies, bins=50)
    ax2.set_xlabel("Entropy")
    ax2.set_ylabel("Frequency")
    plt.show()
```

```
import numpy as np
from scipy.stats import entropy
import matplotlib.pyplot as plt

# Define two probability distributions with non-overlapping
    support sets
p1 = [0.1, 0.2, 0.7]
p2 = [0.8, 0.1, 0.1]
```

```
m = 0.5 * (np.array(p1) + np.array(p2))
jsd = 0.5 * (entropy(p1, m) + entropy(p2, m))
kl1 = entropy(p1, p2)
kl2 = entropy(p2, p1)

print("Jensen-Shannon Divergence: ", jsd)
print("KL Divergence (p1 || p2): ", kl1)
print("KL Divergence (p2 || p1): ", kl2)

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10,4))
ax1.bar(range(len(p1)), p1)
ax1.set_title("Probability Distribution 1")
ax1.set_ylim([0, 1])
ax2.bar(range(len(p2)), p2)
ax2.set_title("Probability Distribution 2")
ax2.set_ylim([0, 1])
plt.show()
```