# Loss Functions

## 1. L1 and L2 loss ¶

*L1* and *L2* are two common loss functions in machine learning which are mainly used to minimize the error.

**L1 loss function** are also known as **Least Absolute Deviations** in short **LAD**. **L2 loss function** are also known as **Least square errors** in short **LS**.

Let's get brief of these two

### L1 Loss function

It is used to minimize the error which is the sum of all the absolute differences in between the true value and the predicted value.

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

### L2 Loss Function

It is also used to minimize the error which is the sum of all the squared differences in between the true value and the pedicted value.

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

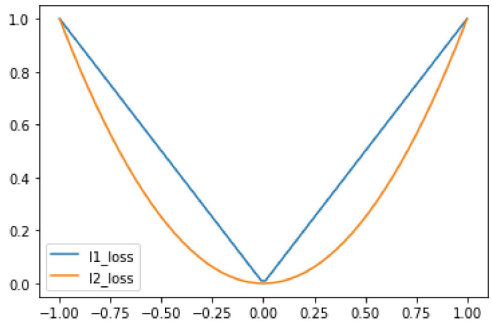
**The disadvantage** of the **L2 norm** is that when there are outliers, these points will account for the main component of the loss. For example, the true value is 1, the prediction is 10 times, the prediction value is 1000 once, and the prediction value of the other times is about 1, obviously the loss value is mainly dominated by 1000.

In [1]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

In [2]:
```
x_guess = tf.lin_space(-1., 1., 100)
x_actual = tf.constant(0,dtype=tf.float32)
```

In [3]:
```
l1_loss = tf.abs((x_guess-x_actual))
l2_loss = tf.square((x_guess-x_actual))
```

In [4]:
```
with tf.Session() as sess:
    x_,l1_,l2_ = sess.run([x_guess, l1_loss, l2_loss])
    plt.plot(x_,l1_,label='l1_loss')
    plt.plot(x_,l2_,label='l2_loss')
    plt.legend()
    plt.show()
```




## 2. Huber Loss

Huber Loss is often used in regression problems. Compared with L2 loss, Huber Loss is less sensitive to outliers(because if the residual is too large, it is a piecewise function, loss is a linear function of the residual).

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Among them, $\delta$ is a set parameter, $y$ represents the real value, and $f(x)$ represents the predicted value.

The advantage of this is that when the residual is small, the loss function is L2 norm, and when the residual is large, it is a linear function of L1 norm

### Pseudo-Huber loss function

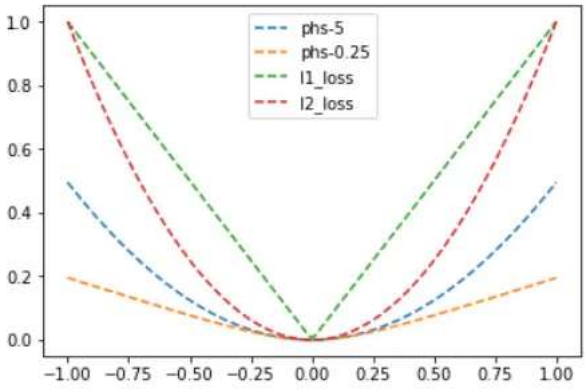
A smooth approximation of Huber loss to ensure that each order is differentiable.

$$L_\delta(a) = \delta^2(\sqrt{1 + (a/\delta)^2} - 1).$$

As such, this function approximates $a^2/2$ for small values of $a$, and approximates a straight line with slope $\delta$ for large values of $a$.

While the above is the most common form, other smooth approximations of the Huber loss function also exist.[5]

Where $\delta$ is the set parameter, the larger the value, the steeper the linear part on both sides.




## 3.Hinge Loss

Hinge loss is often used for binary classification problems, such as ground true: t = 1 or -1, predicted value y = wx + b

In the svm classifier, the definition of hinge loss is

### Hinge loss

From Wikipedia, the free encyclopedia

In machine learning, the **hinge loss** is a loss function used for training classifiers. The hinge loss is used for "maximum-margin" classification, most notably for support vector machines (SVMs).[1] For an intended output $t = \pm 1$ and a classifier score $y$, the hinge loss of the prediction $y$ is defined as

$$\ell(y) = \max(0, 1 - t \cdot y)$$

Note that $y$ should be the "raw" output of the classifier's decision function, not the predicted class label. For instance, in linear SVMs, $y = \mathbf{w} \cdot \mathbf{x} + b$, where $(\mathbf{w}, b)$ are the parameters of the hyperplane and $\mathbf{x}$ is the point to classify.

It can be seen that when $t$ and $y$ have the same sign (meaning $y$ predicts the right class) and $|y| \ge 1$, the hinge loss $\ell(y) = 0$, but when they have opposite sign, $\ell(y)$ increases linearly with $y$ (one-sided error).

In other words, the closer the y is to t, the smaller the loss will be.

In [5]:
```
1  x_guess2 = tf.linspace(-3.,5.,500)
2  x_actual2 = tf.convert_to_tensor([1.]*500)
3
4  #Hinge loss
5  #hinge_loss = tf.losses.hinge_loss(labels=x_actual2, logits=x_guess2)
6  hinge_loss = tf.maximum(0.,1.-(x_guess2*x_actual2))
7  0with tf.Session() as sess:
8      x_,hin_ = sess.run([x_guess2, hinge_loss])
9      plt.plot(x_,hin_,'--', label='hin_')
10     plt.legend()
11     plt.show()
```

```
  File "<ipython-input-5-2caf33f96afd>", line 7
    0with tf.Session() as sess:
     ^
SyntaxError: invalid syntax
```

## 4.Cross-entropy loss

### Cross-entropy loss function and logistic regression [ edit ]

Cross entropy can be used to define a loss function in machine learning and optimization. The true probability $p_i$ is the true label, and the given distribution $q_i$ is the predicted value of the current model.

More specifically, consider logistic regression, which (among other things) can be used to classify observations into two possible classes (often simply labelled $0$ and $1$). The output of the model for a given observation, given a vector of input features $x$, can be interpreted as a probability, which serves as the basis for classifying the observation. The probability is modeled using the logistic function $g(z) = 1/(1+e^{-z})$ where $z$ is some function of the input vector $x$, commonly just a linear function. The probability of the output $y = 1$ is given by

$$q_{y=1} = \hat{y} \equiv g(\mathbf{w}\cdot\mathbf{x}) = 1/(1+e^{-\mathbf{w}\cdot\mathbf{x}}),$$

where the vector of weights $\mathbf{w}$ is optimized through some appropriate algorithm such as gradient descent. Similarly, the complementary probability of finding the output $y = 0$ is simply given by

$$q_{y=0} = 1 - \hat{y}$$

Having set up our notation, $p \in \{y, 1-y\}$ and $q \in \{\hat{y}, 1-\hat{y}\}$, we can use cross entropy to get a measure of dissimilarity between $p$ and $q$:

$$H(p,q) = -\sum_i p_i \log q_i = -y\log\hat{y} - (1-y)\log(1-\hat{y})$$

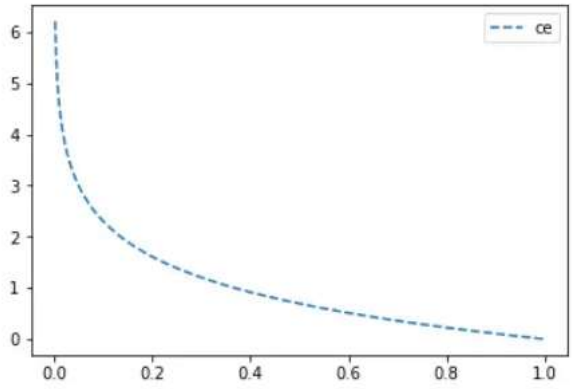
Logistic regression typically optimizes the log loss for all the observations on which it is trained, which is the same as optimizing the average cross-entropy in the sample. For example, suppose we have $N$ samples with each sample indexed by $n = 1, \ldots, N$. The average of the loss function is then given by:

$$J(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log\hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right],$$

where $\hat{y}_n \equiv g(\mathbf{w}\cdot\mathbf{x}_n) = 1/(1+e^{-\mathbf{w}\cdot\mathbf{x}_n})$, with $g(z)$ the logistic function as before.

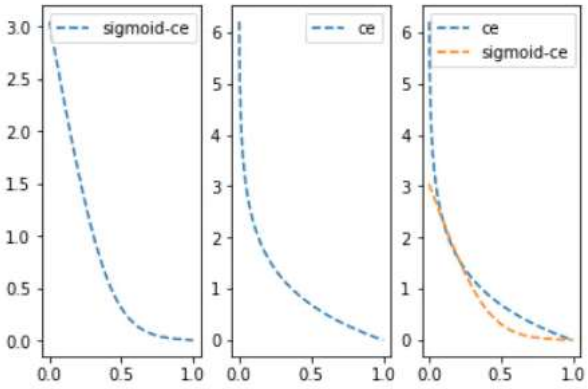
The logistic loss is sometimes called cross-entropy loss. It is also known as log loss (In this case, the binary label is often denoted by {-1,+1}).[2]

The above is mainly to say that cross-entropy loss is mainly applied to binary classification problems. The predicted value is a probability value and the loss is defined according to the cross entropy. Note the value range of the above value: the predicted value of y should be a probability and the value range is [0,1]




## 5.Sigmoid-Cross-entropy loss

The above cross-entropy loss requires that the predicted value is a probability. Generally, we calculate $scores = x * w + b$. Entering this value into the sigmoid function can compress the value range to (0,1).




It can be seen that the sigmoid function smoothes the predicted value(such as directly inputting 0.1 and 0.01 and inputting 0.1, 0.01 sigmoid and then entering, the latter will obviously have a much smaller change value), which makes the predicted value of sigmoid-ce far from the label loss growth is not so steep.

## 6.Softmax cross-entropy loss

First, the softmax function can convert a set of fraction vectors into corresponding probability vectors. Here is the definition of softmax function

---

In mathematics, the **softmax function**, or **normalized exponential function**,[1]:198 is a generalization of the logistic function that "squashes" a $K$-dimensional vector $\mathbf{z}$ of arbitrary real values to a $K$-dimensional vector $\sigma(\mathbf{z})$ of real values in the range (0, 1] that add up to 1. The function is given by

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

---

As above, softmax also implements a vector of 'squashes' k-dimensional real value to the [0,1] range of k-dimensional, while ensuring that the cumulative sum is 1.

According to the definition of cross entropy, probability is required as input.Sigmoid-cross-entropy-loss uses sigmoid to convert the score vector into a probability vector, and softmax-cross-entropy-loss uses a softmax function to convert the score vector into a probability vector.

According to the definition of cross entropy loss.

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

where $p(x)$ represents the probability that classification $x$ is a correct classification, and the value of $p$ can only be 0 or 1. This is the prior value

$q(x)$ is the prediction probability that the $x$ category is a correct classification, and the value range is (0,1)

So specific to a classification problem with a total of C types, then $p(x_j)$, $(0 <_= j <_= C)$ must be only 1 and C-1 is 0(because there can be only one correct classification, correct the probability of classification as correct classification is 1, and the probability of the remaining classification as correct classification is 0)

Then the definition of softmax-cross-entropy-loss can be derived naturally.

Here is the definition of softmax-cross-entropy-loss.

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

Where $f_j$ is the score of all possible categories, and $f_{y_i}$ is the score of ground true class

In [ ]:
```
1
```