Потери

• BinaryCrossEntrpy

Y_true = [1, 1, 0]
Y_pred = [0.9, 0.75, 0.7]

$$BCE = -\frac{1}{N}\sum_{i=1}^{i=N}[y_i \cdot log(\widehat{y_i}) + (1 - y_i) \cdot log(1 - \widehat{y_i})]$$

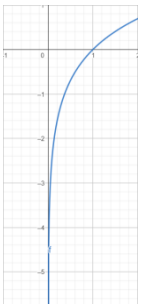
$$-\frac{1}{3}(\,y_1 \cdot \log(\hat{y}_1) + (1 - y_1) \cdot \log(1 - \hat{y}_1) \;+\; y_2 \cdot \log(\hat{y}_2) + (1 - y_2) \cdot \log(1 - \hat{y}_2) \;+\; y_3 \cdot \log(\hat{y}_3) + (1 - y_3) \cdot \log(1 - \hat{y}_3))$$

$$-\frac{1}{3}(\,1 \cdot \log(0.9) + (1 - 1) \cdot \log(1 - 0.9) \;+\; 1 \cdot \log(0.75) + (1 - 1) \cdot \log(1 - 0.75) \;+\; 0 \cdot \log(0.7) + (1 - 0) \cdot \log(1 - 0.7))$$

$$-\frac{1}{3}(\log(0.9) \;+\; \log(0.75) + \log(1 - 0.7))$$

$$-\frac{1}{3}(-0.10536051 \;-\; 0.28768207 \;-\; 1.20397280) \;=\; 0.53233846$$

При попытке вычислить всё вручную можно заметить, что половина значений зануляется.



Вспомним график логарифма от 0 до 1.

Чтобы при правильном предсказании ошибка была равна 0, логарифм должен быть от 1. Поэтому если правильным является значение 1 (y=1), то мы его так и оставляем log(y). Если правильным значением является 0, то мы берём log(1-y), тогда при y=0 будет log(1). Таким образом мы считаем ошибки относительно обоих меток.
Но поскольку нам нужна только одна метка, вторую мы должны занулить. Как? Умножив на 0. Таким образом, если правильная метка 1, то мы делаем 1-y, если 0, то оставляем y.

```
import tensorflow as tf

cross_entropy = tf.keras.losses.BinaryCrossentropy()
loss = cross_entropy([1,1,0], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(0.53233826, shape=(), dtype=float32)
```

У бинарной кросс-энтропи есть параметр: from_logits.

Он отвечает за приведение входных данных к нужному виду. Дело в том, что эта функция ожидает на вход числа от 0 до 1. Но в зависимости от структуры нейронки она может отправить и данные, выходящие за этот диапазон.

Этот параметр отвечает за то, чтобы привести все входные данные к диапазону [0;1] при помощи сигмоиды.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Y_true = [1, 1, 0]
Raw_y_pred = [15, 1, 0]
Y_pred = sigmoid(raw_y_pred) = [0.99999969, 0.73105857, 0.5]

$$-\frac{1}{3}(\log(0.99999969) + \log(0.73105857) + \log(1 - 0.5)) = 0.33546972$$

```
import tensorflow as tf

cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
loss = cross_entropy([1,1,0], [18, 1, 0.])
print(loss)
```

```
tf.Tensor(0.33546963, shape=(), dtype=float32)
```

• BinaryFocalCrossentropy

Это как обычная кросс-энтропия, но логарифм умножается на специальный множитель.

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \quad \text{где} \quad p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise,} \end{cases}$$

Итого выведенная формула выглядит так:

$$-\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \boldsymbol{\alpha} \cdot (\mathbf{1}-\hat{y}_i)^{\gamma} \cdot \log(\hat{y}_i) + (1-y_i) \cdot \boldsymbol{\alpha} \cdot \hat{\boldsymbol{y}}_i^{\gamma} \cdot \log(1-\hat{y}_i)]$$

Y_true = [1, 1, 0]
Y_pred = [0.9, 0.75, 0.7]
Gamma=2
Aplha = 1 // В tensorflow не влияет на результат

$$-\frac{1}{3}(\,(1-\hat{y}_1)^{\gamma} \cdot \log(\hat{y}_1) + (1-\hat{y}_1)^{\gamma} \cdot \log(\hat{y}_1) + \hat{y}_1{}^{\gamma} \cdot \log(1-\hat{y}_1))$$

$$-\frac{1}{3}(\,(1-0.9)^2 \cdot \log(0.9) + (1-0.75)^2 \cdot \log(0.75) + 0.7^2 \cdot \log(0.3))$$

$$-\frac{1}{3}(\,\mathbf{0.01} \cdot (-0.10536051) + \mathbf{0.0625} \cdot (-0.28768207) + \mathbf{0.49} \cdot (-1.20397280))$$
$$= 0.20299346$$

Идея этого подхода в том, чтобы *фокусироваться* на тех значениях, которые являются более ошибочными. Как видно в примере, вы возводим ошибку в степень, таким образом крупные ошибки почти не изменяются (0.49), а мелкие сильно уменьшаются (0.01).

```
import tensorflow as tf

cross_entropy = tf.keras.losses.BinaryFocalCrossentropy(gamma=2)
loss = cross_entropy([1,1,0], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(0.20299341, shape=(), dtype=float32)
```

Также имеется from_logits, потому что эта функция также работает с промежутком (0;1).

• CategoricalCrossentropy

В отличии от бинарной классификации, эта используется только для векторов в OHE. Т.е. вектора [1,0,0], [0,1,0], [0,0,1] будут верными, а вот для вектора [1,1,0] нужно использовать бинарную кросс-энтропию. Конечно, никто не запретит отправить сюда любые данные, но это будет менее эффективно.

Это обычная кросс-энтропия, но предсказанные данные проходят через активатор softmax.

$\hat{y}_k = \frac{e_k}{\sum_{i=1}^{N} e_i}$, где $\hat{y}_k$ - конечный элемент y_pred, e - все элементы из raw_y_pred

Т.е. для вектора [1,2,4] softmax сделает следующее:

$y_1 = \frac{e_1}{\sum_{i=1}^{N} e_i} = \frac{1}{1+2+4} = \frac{1}{7}$; $y_2 = \frac{e_2}{\sum_{i=1}^{N} e_i} = \frac{2}{1+2+4} = \frac{2}{7}$; $y_3 = \frac{e_3}{\sum_{i=1}^{N} e_i} = \frac{4}{1+2+4} = \frac{4}{7}$

Обращаю внимание, что softmax делает так, что сумма всех элементов конечного вектора равна единице. Т.е. он превращает входной вктор в вектор вероятностей.

Для многоклассовой классификации вектор [1,1,0] значит, что объект принадлежит первом и второму классу, но если мы применим softmax, то получим [0.5, 0.5, 0], а это ни туда, ни сюда.

Но если входной вектор равен [1,0,0], то он останется таким же (даже с оговоркой, что нейронки не дают точных чисел, а то-то близкое).

Y_true = [1, 1, 0] // в этом примере вектор не OHE только для того, чтобы показать вычисления
Raw_y_pred = [0.9, 0.75, 0.7]
Y_pred = Raw_y_pred/sum(Raw_y_pred) = [0.38297, 0.31914, 0.29787]

$$-\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i)$$

$$-(1 \cdot \log(0.38297) + 1 \cdot \log(0.31914) + 0 \cdot \log(0.29787))$$
$$= -(-0.95979 - 1.14212 - 1.21109) = 2.10192$$

```
import tensorflow as tf

hinge = tf.keras.losses.CategoricalCrossentropy()
loss = hinge([1,1,0], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(2.1018732, shape=(), dtype=float32)
```

Как видно, формула учитывает только те случаи, где y=1, не добавляя ошибку за несовпадение по 0. Это достигается за счёт OHE и softmax. Softmax делает так, что при увеличении одного признака остальные уменьшаются. А OHE гарантирует, что такой признак будет единственный.

Таким образом эта ошибка будет минимальна, когда единственная единица совпадёт. А это случится только если все остальные значения занулятся.

Также tensorflow выбросит ошибку, если размер массива равен 1, потому что нужно хотя бы 2 класса для классификации.

```
import tensorflow as tf

cce = tf.keras.losses.CategoricalCrossentropy()
loss = cce([1], [0.9])
print(loss)
```

```
tf.Tensor(1.192093e-07, shape=(), dtype=float32)
```

```
D:\programs\python\Lib\site-packages\tensorflow\python\util\dispatch.py:1176: SyntaxWarning: In loss categorical_crossentropy,
expected y_pred.shape to be (batch_size, num_classes) with num_classes > 1. Received: y_pred.shape=(1,). Consider using 'binary
_crossentropy' if you only have 2 classes.
  return dispatch_target(*args, **kwargs)
```

• CategoricalFocalCrossentropy

Уже знаем, что значит Focal. Добавляем фокусирующий множитель.

$$-\sum_{i=0}^{N} y_i \cdot {p_i}^{\gamma} \cdot log(\hat{y}_i), \quad \text{где } p_i = \begin{cases} \hat{y}_i & when\ y_i = 1 \\ 1 - \hat{y}_i & otherwise \end{cases}$$

Y_true = [1, 1, 0]
Raw_y_pred = [0.9, 0.75, 0.7]
Y_pred = Raw_y_pred/sum(Raw_y_pred) = [0.38297, 0.31914, 0.29787]
Gamma=2

$$-(1 \cdot 0.38297^2 \cdot \log(0.38297) + 1 \cdot 0.31914^2 \cdot \log(0.31914) + 0 \cdot 0.29787^2 \cdot \log(0.29787))$$

- CategoricalHinge

- CosineSimilarity

• Hinge

Шарнирная функция. Используется для SVM.

$$\frac{1}{\mathrm{N}}\sum_{i=1}^{N} max(0, \quad 1 - y_i \cdot \hat{y}_i)$$

Эта функция ожидает на вход значения из промежутка [-1;1], поэтому если придёт диапазон [0;1], он будет расширен.

Raw_y_true = [1, 1, 0]
Y_true = Raw_y_true*2-1 = [1, 1, -1]
Y_pred = [0.9, 0.75, 0.7]

$$\frac{1}{3}(max(0, \quad 1 - y_1 \cdot \hat{y}_1) + max(0, \quad 1 - y_2 \cdot \hat{y}_2) + max(0, \quad 1 - y_3 \cdot \hat{y}_3))$$

$$\frac{1}{3}(max(0, \quad 1 - 1 \cdot 0.9) + max(0, \quad 1 - 1 \cdot 0.75) + max(0, \quad 1 + 1 \cdot 0.7))$$

$$\frac{1}{3}(max(0, \ 0.1) + max(0, \ 0.25) + max(0, \ 1.7)) = \frac{1}{3}(0.1 + \ 0.25 + 1.7) = 0.683333$$

```python
import tensorflow as tf

hinge = tf.keras.losses.Hinge()
loss = hinge([1,1,0], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(0.6833334, shape=(), dtype=float32)
```

```python
import tensorflow as tf

hinge = tf.keras.losses.Hinge()
loss = hinge([1,1,-1], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(0.6833334, shape=(), dtype=float32)
```

• SquaredHinge

Возводит максимумы в квадрат. Идея такая же как и везде: ослабить мелкие ошибки и усилить крупные.

$$\frac{1}{\mathrm{N}}\sum_{i=1}^{N} max(0, \quad 1 - y_i \cdot \hat{y}_i)^2$$

$$\frac{1}{3}(max(0, \quad 0.1)^2 + max(0, \quad 0.25)^2 + max(0, \quad 1.7)^2) = \frac{1}{3}(0.01 + \ 0.0625 + 2.89) = 0.9875$$

```python
import tensorflow as tf

hinge = tf.keras.losses.SquaredHinge()
loss = hinge([1,1,0], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(0.9875, shape=(), dtype=float32)
```

• Huber

Функция потерь Хьюбера

Используемая в устойчивой регрессии, которая менее чувствительна к выбросам, чем квадратичная ошибка.

$$\frac{1}{N}\sum_{i=1}^{N}\begin{cases}\frac{a_i^2}{2} & for\ |a_i| \le \delta \\ \delta \cdot \left(|a_i| - \frac{\delta}{2}\right) & otherwise\end{cases} \quad \text{, где } a_i = y_i - \hat{y}_i$$

Y_true = [1, 1, 0]
Y_pred = [0.9, 0.75, 0.7]
Delta=0.5

a1 = 1-0.9 = 0.1
a2 = 1-0.75 = 0.25
a3 = 0-0.7 = -0.7

$$\frac{1}{3}\left(\frac{0.1^2}{2} + \frac{0.25^2}{2} + 0.5 \cdot \left(|-0.7| - \frac{0.5}{2}\right)\right) = \frac{1}{3}(0.005 + 0.03125 + 0.225) = 0.08708333$$

```
import tensorflow as tf

huber = tf.keras.losses.Huber(delta=0.5)
loss = huber([1,1,0], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(0.08708333, shape=(), dtype=float32)
```

• KLDivergence

Функция Кульбака - Лейблера

$$\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log\left(\frac{y_i}{\hat{y}_i}\right)$$

- LogCosh

• MeanAbsoluteError

Самая простая ошибка - среднее арифметическое между отклонениями между метками.

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Y_true = [1, 1, 0]
Y_pred = [0.9, 0.75, 0.7]

$$\frac{1}{3}(|1-0.9| + |1-0.75| + |0-0.7|) = \frac{1}{3}(0.1 + 0.25 + 0.7) = 0.35$$

```
import tensorflow as tf

error = tf.keras.losses.MeanAbsoluteError()
loss = error([1,1,0], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(0.35, shape=(), dtype=float32)
```

• MeanAbsolutePercentageError

$$\frac{100}{\mathrm{N}}\sum_{i=0}^{N}\frac{y_i - \hat{y}_i}{y_i}$$

Можно заметить, что знаменатель может быть равен 0. Чтобы этого избежать, к нему добавляется небольшой эпсилон. Экспериментально было выявлено, что этот эпсилон равен 10^{-7}.

Подсчёта не будет, потому что, если есть y=0, то значения получаются очень большие и иногда даже падает ошибка, что невозможно сохранить число в переменную.

• MeanSquaredError

Средняя квадратичная ошибка. Как абсолютная, но в квадрате. Цель как и везде - ослабить слабые и выделить сильные отклонения.

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$\frac{1}{3}((1-0.9)^2 + (1-0.75)^2 + (0-0.7)^2) = \frac{1}{3}(0.01 + 0.0625 + 0.49) = 0.1875$$

```
import tensorflow as tf

error = tf.keras.losses.MeanSquaredError()
loss = error([1,1,0], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(0.1875, shape=(), dtype=float32)
```

• MeanSquaredLogarithmicError

$$\frac{1}{\mathrm{N}}\sum_{i=0}^{N}(log(1+y_i) - log(1+\hat{y}_i))^2$$

Y_true = [1, 1, 0]
Y_pred = [0.9, 0.75, 0.7]

$$\frac{1}{3}((log(1+1) - log(1+0.9))^2 + (log(1+1) - log(1+0.75))^2 + (log(1+0) - log(1+0.7))^2) = \frac{1}{3}((log(2) - log(1.9))^2 + (log(2) - log(1.75))^2 + (log(1) - log(1.7))^2) = \frac{1}{3}((0.69314 - 0.64185)^2 + (0.69314 - 0.55961)^2 + (0 - 0.53062)^2) = \frac{1}{3}\left(0.05129^2 + 0.13353^2 + (-0.53062)^2\right) = 0.10067283$$

```
import tensorflow as tf

error = tf.keras.losses.MeanSquaredLogarithmicError()
loss = error([1,1,0], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(0.10067596, shape=(), dtype=float32)
```

• Poisson

$$-\frac{1}{N}\sum_{i=1}^{N}\hat{y}_i - y_i \cdot log(\hat{y}_i)$$

Y_true = [1, 1, 0]
Y_pred = [0.9, 0.75, 0.7]

$$-\frac{1}{3}(0.9 - 1 \cdot \log(0.9) + 0.75 - 1 \cdot \log(0.75) + 0.7 - 0 \cdot \log(0.7))$$

$$-\frac{1}{3}(0.9 - (-0.10536) + 0.75 - (-0.28768) + 0.7 - 0 \cdot (-0.35667))$$

$$-\frac{1}{3}(1.00536 + 1.03768)$$

```
import tensorflow as tf

error = tf.keras.losses.Poisson()
loss = error([1,1,0], [0.9, 0.75, 0.7])
print(loss)
```

```
tf.Tensor(0.9143474, shape=(), dtype=float32)
```

- SparseCategoricalCrossentropy