متد add\_metric در تنسورفلو (TensorFlow/Keras) ابزاریه که بهت اجازه می‌ده معیارهای (Metrics) سفارشی رو به مدل اضافه کنی تا عملکرد مدل رو حین آموزش (مثل model.fit) رصد کنی. این متد بر خلاف add\_loss که به ضرر کل مدل اضافه می‌شه، فقط برای نمایش و مانیتورینگ استفاده می‌شه و تو فرآیند بهینه‌سازی تأثیر مستقیم نداره. بیایم این متد رو به‌صورت جامع و ساده توضیح بدم.

**متد add\_metric چیست؟**

add\_metric یه متد تو کلاس‌های tf.keras.layers.Layer و tf.keras.Model هست که بهت اجازه می‌ده معیارهای اضافی (مثل دقت، خطا یا یه مقدار سفارشی) رو تعریف کنی و تو خروجی‌های آموزش (مثل لاج‌ها یا نمودارها) نمایش بدی. این معیارها به لیست model.metrics اضافه می‌شن و تو هر ایپاک یا بچ، مقدارشون به‌روزرسانی و گزارش می‌شه.

**دلیل پیدایش**

* **رصد بهتر مدل**: گاهی تابع ضرر اصلی (مثل mse) به‌تنهایی کافی نیست و می‌خوای معیارهای دیگه‌ای مثل Precision یا Recall رو بررسی کنی.
* **انعطاف‌پذیری**: برای پروژه‌هایی که نیاز به معیارهای خاص دارن (مثلاً تو پزشکی یا مالی)، می‌تونی معیارهای سفارشی تعریف کنی.
* **مانیتورینگ بدون تأثیر روی بهینه‌سازی**: برخلاف add\_loss، این متد فقط برای نمایش استفاده می‌شه و اپتیمایزر رو تغییر نمی‌ده.

**نحوه کار**

وقتی از add\_metric استفاده می‌کنی، یه مقدار (معمولاً یه تنسور اسکالر) به‌عنوان معیار به مدل اضافه می‌شه. این مقدار تو هر به‌روزرسانی (مثل update\_state) تغییر می‌کنه و تو result نمایش داده می‌شه. معیارها تو خروجی model.fit (مثل accuracy: 0.95) ظاهر می‌شن.

**نحوه استفاده**

**1. استفاده تو لایه‌های سفارشی**

می‌تونی تو یه لایه سفارشی معیار اضافه کنی:

import tensorflow as tf

class CustomLayer(tf.keras.layers.Layer):

def \_\_init\_\_(self, units):

super(CustomLayer, self).\_\_init\_\_()

self.units = units

self.dense = tf.keras.layers.Dense(units)

def call(self, inputs):

output = self.dense(inputs)

*# معیار سفارشی: میانگین خروجی‌ها*

mean\_output = tf.reduce\_mean(output)

self.add\_metric(mean\_output, name='mean\_output', aggregation='mean')

return output

*# استفاده تو مدل*

model = tf.keras.Sequential([

CustomLayer(10),

tf.keras.layers.Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy')

model.fit(tf.random.uniform((100, 5)), tf.random.uniform((100, 1), 0, 2, dtype=tf.int32), epochs=5)

**2. استفاده تو مدل‌های سفارشی**

می‌تونی تو یه مدل سفارشی معیار اضافه کنی:

class CustomModel(tf.keras.Model):

def \_\_init\_\_(self):

super(CustomModel, self).\_\_init\_\_()

self.layer1 = tf.keras.layers.Dense(10, activation='relu')

self.layer2 = tf.keras.layers.Dense(1, activation='sigmoid')

def call(self, inputs):

x = self.layer1(inputs)

output = self.layer2(x)

*# معیار سفارشی: واریانس خروجی لایه اول*

variance = tf.reduce\_mean(tf.square(x - tf.reduce\_mean(x)))

self.add\_metric(variance, name='output\_variance', aggregation='mean')

return output

*# استفاده از مدل*

model = CustomModel()

model.compile(optimizer='adam', loss='binary\_crossentropy')

model.fit(tf.random.uniform((100, 5)), tf.random.uniform((100, 1), 0, 2, dtype=tf.int32), epochs=5)

**3. استفاده مستقیم تو مدل**

اگه از مدل Sequential استفاده می‌کنی، می‌تونی معیار رو مستقیماً اضافه کنی:

model = tf.keras.Sequential([

tf.keras.layers.Dense(10, input\_dim=5, activation='relu'),

tf.keras.layers.Dense(1, activation='sigmoid')

])

*# معیار سفارشی برای خروجی‌ها*

model.add\_metric(tf.reduce\_mean(model.output), name='mean\_output', aggregation='mean')

model.compile(optimizer='adam', loss='binary\_crossentropy')

model.fit(tf.random.uniform((100, 5)), tf.random.uniform((100, 1), 0, 2, dtype=tf.int32), epochs=5)

**کاربردها تو صنعت و حرفه**

1. **پزشکی**: معیارهایی مثل Sensitivity یا Specificity رو برای تشخیص بیماری‌ها رصد کن.
2. **مالی**: واریانس پیش‌بینی‌ها رو با add\_metric مانیتور کن تا از نوسانات غیرمنتظره جلوگیری کنی.
3. **بازاریابی**: نرخ کلیک (Click-Through Rate) سفارشی رو برای سیستم‌های پیشنهاددهنده دنبال کن.
4. **رباتیک**: دقت حرکت رو با معیارهای خاص تو زمان واقعی بررسی کن.

**عملکرد تو کار**

* **مانیتورینگ**: معیارها تو هر ایپاک یا بچ گزارش می‌شن و بهت کمک می‌کنن روند آموزش رو تحلیل کنی.
* **بدون تأثیر روی بهینه‌سازی**: فقط برای نمایش هستن و اپتیمایزر رو تغییر نمی‌دن.
* **انعطاف‌پذیری**: می‌تونی هر نوع محاسبه‌ای (مثل میانگین، واریانس، یا حتی یه شرط خاص) رو به‌عنوان معیار تعریف کنی.

**نکات تکمیلی**

* **شکل معیار**: معیار باید یه تنسور اسکالر باشه (با tf.reduce\_mean یا tf.reduce\_sum می‌تونی به اسکالر تبدیلش کنی).
* **aggregation**: پارامتر aggregation (مثل 'mean' یا 'sum') مشخص می‌کنه چطور مقادیر رو جمع‌بندی کنه (پیش‌فرض 'mean' است).
* **پشتیبانی از وزن‌دهی**: اگه sample\_weight تو model.fit داری، معیار به‌صورت خودکار وزن‌دهی می‌شه.
* **عملکرد محاسباتی**: استفاده زیاد از معیارهای پیچیده می‌تونه زمان آموزش رو کمی افزایش بده.

**ترکیب با سایر روش‌ها**

* **با add\_loss**: می‌تونی ضرر و معیار رو همزمان استفاده کنی؛ مثلاً ضرر برای بهینه‌سازی و معیار برای رصد.
* **با Callback‌ها**: با Callback مثل ModelCheckpoint، معیارها رو برای ذخیره مدل استفاده کن.
* **با متریک‌های پیش‌فرض**: معیارهای سفارشی رو با متریک‌های آماده (مثل accuracy) ترکیب کن.

**مقایسه با سایر متدها**

* **در مقابل add\_loss**: add\_loss ضرر رو به بهینه‌سازی اضافه می‌کنه، ولی add\_metric فقط نمایش می‌ده.
* **در مقابل متریک‌های پیش‌فرض**: متریک‌های آماده (مثل Precision) محدودن، ولی add\_metric بهت آزادی کامل می‌ده.
* **در مقابل لاج دستی**: به‌جای چاپ دستی با print، add\_metric معیارها رو تو خروجی استاندارد Keras نشون می‌ده.

**مثال‌های ساده و عملی**

**مثال 1: معیار میانگین خروجی**

class OutputMeanLayer(tf.keras.layers.Layer):

def \_\_init\_\_(self, units):

super().\_\_init\_\_()

self.dense = tf.keras.layers.Dense(units)

def call(self, inputs):

output = self.dense(inputs)

self.add\_metric(tf.reduce\_mean(output), name='mean\_output')

return output

model = tf.keras.Sequential([

OutputMeanLayer(10),

tf.keras.layers.Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy')

model.fit(tf.random.uniform((100, 5)), tf.random.uniform((100, 1), 0, 2, dtype=tf.int32), epochs=5)

**مثال 2: معیار واریانس**

class VarianceModel(tf.keras.Model):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.dense1 = tf.keras.layers.Dense(10, activation='relu')

self.dense2 = tf.keras.layers.Dense(1, activation='sigmoid')

def call(self, inputs):

x = self.dense1(inputs)

output = self.dense2(x)

variance = tf.reduce\_mean(tf.square(x - tf.reduce\_mean(x)))

self.add\_metric(variance, name='layer1\_variance')

return output

model = VarianceModel()

model.compile(optimizer='adam', loss='binary\_crossentropy')

model.fit(tf.random.uniform((100, 5)), tf.random.uniform((100, 1), 0, 2, dtype=tf.int32), epochs=5)

**مثال تخصصی برای بازار کار**

تو یه سیستم تشخیص تقلب، می‌تونی معیار سفارشی برای رصد تعداد پیش‌بینی‌های با احتمال بالا بسازی:

class FraudAlertModel(tf.keras.Model):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.dense1 = tf.keras.layers.Dense(20, activation='relu')

self.dense2 = tf.keras.layers.Dense(1, activation='sigmoid')

def call(self, inputs):

x = self.dense1(inputs)

output = self.dense2(x)

*# معیار: تعداد پیش‌بینی‌های با احتمال بالای 0.9*

high\_confidence\_count = tf.reduce\_sum(tf.cast(output > 0.9, tf.float32))

self.add\_metric(high\_confidence\_count, name='high\_confidence\_count')

return output

model = FraudAlertModel()

model.compile(optimizer='adam', loss='binary\_crossentropy')

model.fit(tf.random.uniform((100, 5)), tf.random.uniform((100, 1), 0, 2, dtype=tf.int32), epochs=5)

این معیار می‌تونه به تحلیل‌گرها کمک کنه تعداد پیش‌بینی‌های مطمئن رو تو سیستم بانکی رصد کنن.

**پیشنهادها**

* **تست معیارهای مختلف**: معیارهای متفاوتی رو با add\_metric امتحان کن و بهترین رو انتخاب کن.
* **نمایش بصری**: از TensorBoard برای رصد معیارها استفاده کن.
* **ترکیب با داده‌های واقعی**: معیارها رو با داده‌های واقعی تست کن تا عملکرد واقعی مدل رو ببینی.

**نتیجه‌گیری**

متد add\_metric ابزاری عالی برای رصد و تحلیل مدل‌های تنسورفلوئه که بهت اجازه می‌ده معیارهای سفارشی رو به‌راحتی تعریف و مانیتور کنی. این متد بدون تأثیر روی بهینه‌سازی، دید بهتری از روند آموزش بهت می‌ده و تو پروژه‌های صنعتی و تحقیقاتی خیلی کاربرد داره.