The targets are already ranked, all you need to do is rank your predictions and then compute Pearson's correlation coefficient between your ranked predictions and the targets. It might be instructive to look at this block of code in the example model. Here's a quick and dirty snippet of code that does the same thing with Tensorflow.

import tensorflow as tf import numpy as np import pandas as pd

def correlation(predictions, targets): """ From: https://github.com/numerai/example-scripts/blob/master/example_model.py#L21 """ if not isinstance(predictions, pd.Series): predictions = pd.Series(predictions) ranked_preds = predictions.rank(pct=True, method="first") return np.corrcoef(ranked_preds, targets)[0, 1]

def corrcoef(x, y): """ np.corrcoef() implemented with tf primitives """ mx = tf.math.reduce_mean(x) my = tf.math.reduce_mean(y) xm, ym = x - mx, y - my r_num = tf.math.reduce_sum(xm * ym) r_den = tf.norm(xm) * tf.norm(ym) return r_num / (r_den + tf.keras.backend.epsilon())

def tf_correlation(predictions, targets): ranked_preds = tf.cast(tf.argsort(tf.argsort(predictions, stable=True)), targets.dtype) return corrcoef(ranked_preds, targets)

targets = np.array([0.0, 0.25, 0.5, 0.75, 1.0], dtype=np.float32) predictions = np.random.rand(targets.shape[0])

print(correlation(predictions, targets)) print(tf_correlation(tf.convert_to_tensor(predictions), tf.convert_to_tensor(targets)))

One more thing: The tf_correlation

function isn't differentiable, which makes it rather useless if you want to use it in a loss function. Also, the code above was quite hastily written in response to your post (although, I did run it in a notebook to verify that np

and tf

return the same number), so you might want to review it a couple of times before using it.