Searching for the Higgs Boson[¶](https://www.kaggle.com/code/ryanholbrook/detecting-the-higgs-boson-with-tpus" \l "Searching-for-the-Higgs-Boson" \t "_self)

Standart Model, doğanın en temel kuvvetlerinden bazılarını tanımlayan bir parçacık fiziği teorisidir. Temel parçacıklardan biri olan Higgs bozonu, maddenin **kütlesinden** sorumludur. İlk olarak 1964'te teorize edilen Higgs bozonu, yaklaşık elli yıl boyunca gözlemden kaçtı. 2012'de ise Büyük Hadron Çarpıştırıcısı'nda deneysel olarak nihayet gözlemlendi. Bu deneyler, milyonlarca gigabayt veri üretti.

Bunlar gibi büyük ve karmaşık veri setleri, derin öğrenmenin üstün geldiği yerlerdir. Bu notebook'ta (çalışma defterinde), gözlemlenen bir parçacık çarpışmasının bir Higgs bozonu üretip üretmediğini belirlemek için Geniş ve Derin (Wide and Deep) bir sinir ağı kuracağız.

# The Collision Data[¶](https://www.kaggle.com/code/ryanholbrook/detecting-the-higgs-boson-with-tpus#The-Collision-Data)

Yüksek enerjili protonların çarpışması, Higgs bozonu gibi yeni parçacıklar üretebilir. Ancak bu parçacıklar neredeyse anında bozundukları için doğrudan gözlemlenemezler. Bu nedenle, yeni bir parçacığın varlığını tespit etmek için, bozundukları parçacıkların, yani "bozunma ürünlerinin" davranışlarını gözlemleriz.

Higgs veri kümesi, bozunma ürünlerinin 21 "düşük seviyeli" özelliğini ve bunlardan türetilen 7 "yüksek seviyeli" özelliği daha içerir.

# Wide and Deep Neural Networks[¶](https://www.kaggle.com/code/ryanholbrook/detecting-the-higgs-boson-with-tpus#Wide-and-Deep-Neural-Networks)

**Geniş ve Derin (Wide and Deep)** bir ağ, doğrusal bir katmanı derin bir yoğun katman yığınıyla yan yana eğitir. Geniş ve Derin ağlar, genellikle tablo şeklindeki veri kümelerinde etkilidir.

Hem veri seti hem de model, kursta kullandıklarımızdan çok daha büyüktür. Eğitimi hızlandırmak için, büyük iş yükleri için ideal bir hızlandırıcı olan Kaggle'ın **Tensor İşleme Birimlerini (TPU)** kullanacağız.

Deney yapmayı kolaylaştırmak için bazı hiperparametreleri buraya topladık. Kendiniz denemek için bu not defterini **buraya tıklayarak** kopyalayın!

*# Model Configuration*

UNITS = 2 \*\* 11 *# 2048*

ACTIVATION = 'relu'

DROPOUT = 0.1

*# Training Configuration*

BATCH\_SIZE\_PER\_REPLICA = 2 \*\* 11 *# powers of 128 are best*

Sonraki birkaç bölümde TPU hesaplaması, veri hattı ve sinir ağı modeli anlatılmaktadır. Sadece sonuçları görmek isterseniz, sonuna geçebilirsiniz!

# Setup[¶](https://www.kaggle.com/code/ryanholbrook/detecting-the-higgs-boson-with-tpus" \l "Setup" \t "_self)

Bu bölüm, içe aktarmalarımıza ek olarak, dizüstü bilgisayarımızı TPU'ya bağlayacak ve bir dağıtım stratejisi oluşturacak bazı kodlar içeriyor. Her TPU'nun bağımsız çalışan sekiz işlem çekirdeği vardır. Bir dağıtım stratejisiyle, işi aralarında nasıl bölüştürmek istediğimizi tanımlarız.

*# TensorFlow*

import tensorflow as tf

print("Tensorflow version " + tf.\_\_version\_\_)

*# Detect and init the TPU*

try: *# detect TPUs*

tpu = tf.distribute.cluster\_resolver.TPUClusterResolver.connect() *# TPU detection*

strategy = tf.distribute.TPUStrategy(tpu)

except **ValueError**: *# detect GPUs*

strategy = tf.distribute.get\_strategy() *# default strategy that works on CPU and single GPU*

print("Number of accelerators: ", strategy.num\_replicas\_in\_sync)

*# Plotting*

import pandas as pd

import matplotlib.pyplot as plt

*# Matplotlib defaults*

plt.style.use('seaborn-whitegrid')

plt.rc('figure', autolayout=True)

plt.rc('axes', labelweight='bold', labelsize='large',

titleweight='bold', titlesize=18, titlepad=10)

*# Data*

from kaggle\_datasets import KaggleDatasets

from tensorflow.io import FixedLenFeature

AUTO = tf.data.experimental.AUTOTUNE

*# Model*

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras import callbacks

Tensorflow version 2.4.1

Number of accelerators: 8

TensorFlow'un artık sekiz hızlandırıcı algıladığını unutmayın. TPU kullanmak, aynı anda sekiz GPU kullanmak gibidir.

# Load Data[¶](https://www.kaggle.com/code/ryanholbrook/detecting-the-higgs-boson-with-tpus#Load-Data)

Veri kümesi, TFRecords adlı ikili bir dosya biçiminde kodlanmıştır. Bu iki fonksiyon, TFRecords'u ayrıştıracak ve eğitim için kullanabileceğimiz bir TensorFlow tf.data.Dataset nesnesi oluşturacaktır.

def make\_decoder(feature\_description):

def decoder(example):

example = tf.io.parse\_single\_example(example, feature\_description)

features = tf.io.parse\_tensor(example['features'], tf.float32)

features = tf.reshape(features, [28])

label = example['label']

return features, label

return decoder

def load\_dataset(filenames, decoder, ordered=False):

AUTO = tf.data.experimental.AUTOTUNE

ignore\_order = tf.data.Options()

if **not** ordered:

ignore\_order.experimental\_deterministic = False

dataset = (

tf.data

.TFRecordDataset(filenames, num\_parallel\_reads=AUTO)

.with\_options(ignore\_order)

.map(decoder, AUTO)

)

return dataset

dataset\_size = int(11e6)

validation\_size = int(5e5)

training\_size = dataset\_size - validation\_size

*# For model.fit*

batch\_size = BATCH\_SIZE\_PER\_REPLICA \* strategy.num\_replicas\_in\_sync

steps\_per\_epoch = training\_size // batch\_size

validation\_steps = validation\_size // batch\_size

*# For model.compile*

steps\_per\_execution = 256

feature\_description = {

'features': FixedLenFeature([], tf.string),

'label': FixedLenFeature([], tf.float32),

}

decoder = make\_decoder(feature\_description)

data\_dir = KaggleDatasets().get\_gcs\_path('higgs-boson')

train\_files = tf.io.gfile.glob(data\_dir + '/training' + '/\*.tfrecord')

valid\_files = tf.io.gfile.glob(data\_dir + '/validation' + '/\*.tfrecord')

ds\_train = load\_dataset(train\_files, decoder, ordered=False)

ds\_train = (

ds\_train

.cache()

.repeat()

.shuffle(2 \*\* 19)

.batch(batch\_size)

.prefetch(AUTO)

)

ds\_valid = load\_dataset(valid\_files, decoder, ordered=False)

ds\_valid = (

ds\_valid

.batch(batch\_size)

.cache()

.prefetch(AUTO)

)**Model**[**¶**](https://www.kaggle.com/code/ryanholbrook/detecting-the-higgs-boson-with-tpus#Model)

Veriler hazır olduğuna göre, ağı tanımlayalım. Ağın derin dalını, derste kullandığımız Sıralı yöntemden biraz daha esnek olan Keras'ın Fonksiyonel API'sini kullanarak tanımlıyoruz.

def dense\_block(units, activation, dropout\_rate, l1=None, l2=None):

def make(inputs):

x = layers.Dense(units)(inputs)

x = layers.BatchNormalization()(x)

x = layers.Activation(activation)(x)

x = layers.Dropout(dropout\_rate)(x)

return x

return make

with strategy.scope():

*# Wide Network*

wide = keras.experimental.LinearModel()

*# Deep Network*

inputs = keras.Input(shape=[28])

x = dense\_block(UNITS, ACTIVATION, DROPOUT)(inputs)

x = dense\_block(UNITS, ACTIVATION, DROPOUT)(x)

x = dense\_block(UNITS, ACTIVATION, DROPOUT)(x)

x = dense\_block(UNITS, ACTIVATION, DROPOUT)(x)

x = dense\_block(UNITS, ACTIVATION, DROPOUT)(x)

outputs = layers.Dense(1)(x)

deep = keras.Model(inputs=inputs, outputs=outputs)

*# Wide and Deep Network*

wide\_and\_deep = keras.experimental.WideDeepModel(

linear\_model=wide,

dnn\_model=deep,

activation='sigmoid',

)

wide\_and\_deep.compile(

loss='binary\_crossentropy',

optimizer='adam',

metrics=['AUC', 'binary\_accuracy'],

experimental\_steps\_per\_execution=steps\_per\_execution,

)

# Training[¶](https://www.kaggle.com/code/ryanholbrook/detecting-the-higgs-boson-with-tpus#Training)

Eğitim sırasında her zamanki gibi EarlyStopping geri aramasını kullanacağız. Ayrıca bir öğrenme oranı çizelgesi tanımladığımızı da unutmayın. Eğitim süresince öğrenme oranını kademeli olarak azaltmanın performansı artırabileceği (ağırlıklar minimum seviyeye "yerleşir") görülmüştür. Bu çizelge, doğrulama kaybı bir dönemden sonra azalmazsa öğrenme oranını 0,2 ile çarpacaktır.

early\_stopping = callbacks.EarlyStopping(

patience=2,

min\_delta=0.001,

restore\_best\_weights=True,

)

lr\_schedule = callbacks.ReduceLROnPlateau(

patience=0,

factor=0.2,

min\_lr=0.001,

)

history = wide\_and\_deep.fit(

ds\_train,

validation\_data=ds\_valid,

epochs=50,

steps\_per\_epoch=steps\_per\_epoch,

validation\_steps=validation\_steps,

callbacks=[early\_stopping, lr\_schedule],

)

Epoch 1/50

640/640 [==============================] - 163s 255ms/step - loss: 0.6049 - auc: 0.7589 - binary\_accuracy: 0.6923 - val\_loss: 0.7150 - val\_auc: 0.6417 - val\_binary\_accuracy: 0.6334

Epoch 2/50

640/640 [==============================] - 28s 44ms/step - loss: 0.4974 - auc: 0.8349 - binary\_accuracy: 0.7525 - val\_loss: 0.5277 - val\_auc: 0.8233 - val\_binary\_accuracy: 0.7428

Epoch 3/50

640/640 [==============================] - 28s 44ms/step - loss: 0.4803 - auc: 0.8476 - binary\_accuracy: 0.7638 - val\_loss: 0.5147 - val\_auc: 0.8315 - val\_binary\_accuracy: 0.7528

Epoch 4/50

640/640 [==============================] - 29s 45ms/step - loss: 0.4712 - auc: 0.8540 - binary\_accuracy: 0.7698 - val\_loss: 0.5038 - val\_auc: 0.8385 - val\_binary\_accuracy: 0.7582

Epoch 5/50

640/640 [==============================] - 28s 44ms/step - loss: 0.4646 - auc: 0.8585 - binary\_accuracy: 0.7738 - val\_loss: 0.5008 - val\_auc: 0.8395 - val\_binary\_accuracy: 0.7591

Epoch 6/50

640/640 [==============================] - 29s 45ms/step - loss: 0.4591 - auc: 0.8622 - binary\_accuracy: 0.7773 - val\_loss: 0.4709 - val\_auc: 0.8606 - val\_binary\_accuracy: 0.7754

Epoch 7/50

640/640 [==============================] - 27s 43ms/step - loss: 0.4537 - auc: 0.8657 - binary\_accuracy: 0.7807 - val\_loss: 0.4631 - val\_auc: 0.8620 - val\_binary\_accuracy: 0.7787

Epoch 8/50

640/640 [==============================] - 27s 43ms/step - loss: 0.4494 - auc: 0.8686 - binary\_accuracy: 0.7836 - val\_loss: 0.4565 - val\_auc: 0.8670 - val\_binary\_accuracy: 0.7828

Epoch 9/50

640/640 [==============================] - 28s 44ms/step - loss: 0.4455 - auc: 0.8711 - binary\_accuracy: 0.7860 - val\_loss: 0.4491 - val\_auc: 0.8713 - val\_binary\_accuracy: 0.7863

Epoch 10/50

640/640 [==============================] - 29s 45ms/step - loss: 0.4414 - auc: 0.8737 - binary\_accuracy: 0.7885 - val\_loss: 0.4431 - val\_auc: 0.8751 - val\_binary\_accuracy: 0.7891

Epoch 11/50

640/640 [==============================] - 28s 43ms/step - loss: 0.4379 - auc: 0.8759 - binary\_accuracy: 0.7908 - val\_loss: 0.4409 - val\_auc: 0.8748 - val\_binary\_accuracy: 0.7896

Epoch 12/50

640/640 [==============================] - 27s 43ms/step - loss: 0.4345 - auc: 0.8780 - binary\_accuracy: 0.7927 - val\_loss: 0.4377 - val\_auc: 0.8787 - val\_binary\_accuracy: 0.7929

Epoch 13/50

640/640 [==============================] - 27s 43ms/step - loss: 0.4311 - auc: 0.8801 - binary\_accuracy: 0.7949 - val\_loss: 0.4325 - val\_auc: 0.8800 - val\_binary\_accuracy: 0.7946

Epoch 14/50

640/640 [==============================] - 27s 43ms/step - loss: 0.4278 - auc: 0.8821 - binary\_accuracy: 0.7970 - val\_loss: 0.4314 - val\_auc: 0.8811 - val\_binary\_accuracy: 0.7959

Epoch 15/50

640/640 [==============================] - 27s 43ms/step - loss: 0.4247 - auc: 0.8840 - binary\_accuracy: 0.7989 - val\_loss: 0.4284 - val\_auc: 0.8820 - val\_binary\_accuracy: 0.7967

Epoch 16/50

640/640 [==============================] - 29s 45ms/step - loss: 0.4214 - auc: 0.8859 - binary\_accuracy: 0.8008 - val\_loss: 0.4268 - val\_auc: 0.8828 - val\_binary\_accuracy: 0.7976

Epoch 17/50

640/640 [==============================] - 29s 46ms/step - loss: 0.4185 - auc: 0.8876 - binary\_accuracy: 0.8024 - val\_loss: 0.4257 - val\_auc: 0.8838 - val\_binary\_accuracy: 0.7988

Epoch 18/50

640/640 [==============================] - 29s 45ms/step - loss: 0.4150 - auc: 0.8897 - binary\_accuracy: 0.8047 - val\_loss: 0.4246 - val\_auc: 0.8841 - val\_binary\_accuracy: 0.7992

Epoch 19/50

640/640 [==============================] - 29s 46ms/step - loss: 0.4118 - auc: 0.8915 - binary\_accuracy: 0.8066 - val\_loss: 0.4249 - val\_auc: 0.8840 - val\_binary\_accuracy: 0.7989

Epoch 20/50

640/640 [==============================] - 29s 45ms/step - loss: 0.4083 - auc: 0.8935 - binary\_accuracy: 0.8086 - val\_loss: 0.4246 - val\_auc: 0.8844 - val\_binary\_accuracy: 0.7995

history\_frame = pd.DataFrame(history.history)

history\_frame.loc[:, ['loss', 'val\_loss']].plot(title='Cross-entropy Loss')

history\_frame.loc[:, ['auc', 'val\_auc']].plot(title='AUC');