Introduction

Bu kursta şimdiye kadar, sinir ağlarının regresyon problemlerini nasıl çözebileceğini öğrendik. Şimdi sinir ağlarını bir diğer yaygın makine öğrenimi problemi olan sınıflandırmaya uygulayacağız. Şimdiye kadar öğrendiğimiz hemen hemen her şey hala geçerli. Temel fark, kullandığımız kayıp fonksiyonunda ve son katmanın üretmesini istediğimiz çıktı türlerinde.

# Binary Classification

İki sınıftan birine sınıflandırma, yaygın bir makine öğrenimi problemidir. Bir müşterinin alışveriş yapma olasılığını, bir kredi kartı işleminin sahte olup olmadığını, derin uzay sinyallerinin yeni bir gezegenin kanıtını gösterip göstermediğini veya bir tıbbi testin bir hastalığın kanıtı olup olmadığını tahmin etmek isteyebilirsiniz. Bunların hepsi **ikili sınıflandırma (binary classification)** problemleridir.

Ham verilerinizde, sınıflar **"Evet"** ve **"Hayır"** veya **"Köpek"** ve **"Kedi"** gibi metinlerle temsil edilebilir. Bu verileri kullanmadan önce bir **sınıf etiketi** atayacağız: bir sınıf **0** olacak, diğeri ise **1** olacaktır. Sayısal etiketler atamak, verileri bir sinir ağının kullanabileceği bir biçime sokar.

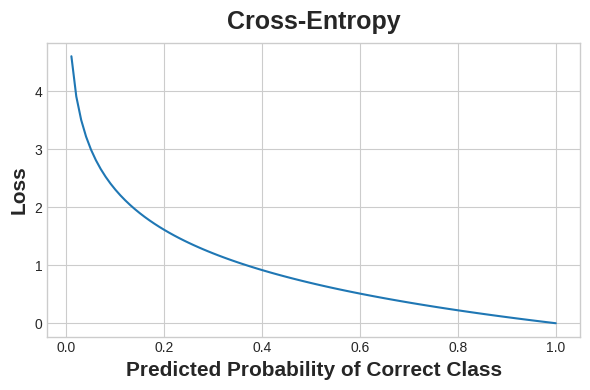
# Accuracy and Cross-Entropy

"Accuracy (doğruluk)", bir sınıflandırma problemindeki başarıyı ölçmek için kullanılan birçok metrikten biridir. Doğruluk, doğru tahminlerin toplam tahminlere oranıdır: **accuracy = number\_correct / total**. Her zaman doğru tahmin yapan bir modelin doğruluk puanı **1.0** olur. Diğer her şey eşit olduğunda, veri setindeki sınıflar yaklaşık olarak aynı sıklıkta meydana geldiğinde doğruluk kullanmak makul bir metriktir.

Doğruluğun (ve diğer çoğu sınıflandırma metriğinin) sorunu, bir kayıp fonksiyonu olarak kullanılamamasıdır. SGD, sorunsuz bir şekilde değişen bir kayıp fonksiyonuna ihtiyaç duyar, ancak sayımların bir oranı olan doğruluk, "sıçramalar" halinde değişir. Bu yüzden, kayıp fonksiyonu olarak hareket etmesi için bir ikame seçmeliyiz. Bu ikame **çapraz entropi (cross-entropy)** fonksiyonudur.

Şimdi, kayıp fonksiyonunun eğitim sırasında ağın **hedefini** tanımladığını hatırlayın. Regresyonda amacımız, beklenen sonuç ile tahmin edilen sonuç arasındaki mesafeyi en aza indirmekti. Bu mesafeyi ölçmek için MAE'yi seçtik.

Sınıflandırma için ise, bunun yerine **olasılıklar** arasında bir mesafe istiyoruz ve çapraz entropi bunu sağlar. **Çapraz entropi**, bir olasılık dağılımından diğerine olan mesafenin bir tür ölçüsüdür.

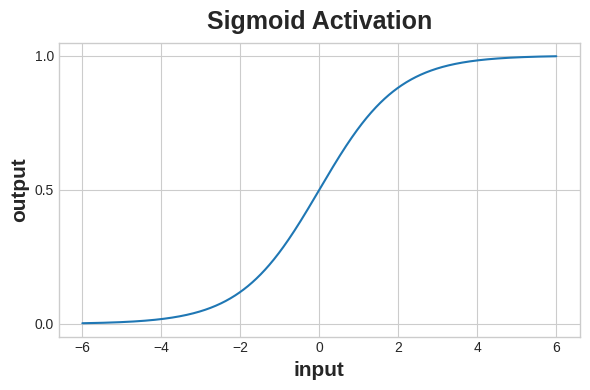


Temel fikir, ağımızın doğru sınıfı **1.0** olasılıkla tahmin etmesini istememizdir. Tahmin edilen olasılık 1.0'dan ne kadar uzaksa, çapraz entropi kaybı da o kadar büyük olacaktır.

Çapraz entropiyi kullanmamızın teknik nedenleri biraz inceliklidir, ancak bu bölümden çıkarılması gereken ana şey şudur: sınıflandırma kaybı için çapraz entropiyi kullanın; önemseyebileceğiniz diğer metrikler (doğruluk gibi) onunla birlikte iyileşme eğilimi gösterecektir.

# Making Probabilities with the Sigmoid Function

Çapraz entropi ve doğruluk fonksiyonlarının her ikisi de girdi olarak olasılıklara, yani 0 ile 1 arasındaki sayılara ihtiyaç duyar. Yoğun bir katman tarafından üretilen gerçek değerli çıktıları olasılıklara dönüştürmek için, sigmoid aktivasyon adı verilen yeni bir aktivasyon fonksiyonu türü ekleriz.



*The sigmoid function maps real numbers into the interval*[0,1]*.*

Nihai sınıf tahminini elde etmek için bir **eşik olasılığı (threshold)** tanımlarız. Genellikle bu 0.5'tir, böylece yuvarlama bize doğru sınıfı verir: 0.5'in altı 0 etiketli sınıfı, 0.5 veya üstü ise 1 etiketli sınıfı ifade eder. Keras'ın **accuracy** metriğinde varsayılan olarak kullandığı eşik 0.5'tir.

# Example - Binary Classification[¶](https://www.kaggle.com/code/ryanholbrook/binary-classification" \l "Example---Binary-Classification" \t "_self)

Şimdi bunu deneyelim!

**İyonosfer (Ionosphere)** veri seti, Dünya atmosferinin iyonosfer katmanına odaklanan radar sinyallerinden elde edilen özellikleri içerir. Buradaki görev, sinyalin bir nesnenin varlığını mı yoksa sadece boş havayı mı gösterdiğini belirlemektir.

import pandas as pd

from IPython.display import display

ion = pd.read\_csv('../input/dl-course-data/ion.csv', index\_col=0)

display(ion.head())

df = ion.copy()

df['Class'] = df['Class'].map({'good': 0, 'bad': 1})

df\_train = df.sample(frac=0.7, random\_state=0)

df\_valid = df.drop(df\_train.index)

max\_ = df\_train.max(axis=0)

min\_ = df\_train.min(axis=0)

df\_train = (df\_train - min\_) / (max\_ - min\_)

df\_valid = (df\_valid - min\_) / (max\_ - min\_)

df\_train.dropna(axis=1, inplace=True) *# drop the empty feature in column 2*

df\_valid.dropna(axis=1, inplace=True)

X\_train = df\_train.drop('Class', axis=1)

X\_valid = df\_valid.drop('Class', axis=1)

y\_train = df\_train['Class']

y\_valid = df\_valid['Class']

| V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10 | ... | V26 | V27 | V28 | V29 | V30 | V31 | V32 | V33 | V34 | Class |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0.99539 | -0.05889 | 0.85243 | 0.02306 | 0.83398 | -0.37708 | 1.00000 | 0.03760 | ... | -0.51171 | 0.41078 | -0.46168 | 0.21266 | -0.34090 | 0.42267 | -0.54487 | 0.18641 | -0.45300 | good |
| 2 | 1 | 0 | 1.00000 | -0.18829 | 0.93035 | -0.36156 | -0.10868 | -0.93597 | 1.00000 | -0.04549 | ... | -0.26569 | -0.20468 | -0.18401 | -0.19040 | -0.11593 | -0.16626 | -0.06288 | -0.13738 | -0.02447 | bad |
| 3 | 1 | 0 | 1.00000 | -0.03365 | 1.00000 | 0.00485 | 1.00000 | -0.12062 | 0.88965 | 0.01198 | ... | -0.40220 | 0.58984 | -0.22145 | 0.43100 | -0.17365 | 0.60436 | -0.24180 | 0.56045 | -0.38238 | good |
| 4 | 1 | 0 | 1.00000 | -0.45161 | 1.00000 | 1.00000 | 0.71216 | -1.00000 | 0.00000 | 0.00000 | ... | 0.90695 | 0.51613 | 1.00000 | 1.00000 | -0.20099 | 0.25682 | 1.00000 | -0.32382 | 1.00000 | bad |
| 5 | 1 | 0 | 1.00000 | -0.02401 | 0.94140 | 0.06531 | 0.92106 | -0.23255 | 0.77152 | -0.16399 | ... | -0.65158 | 0.13290 | -0.53206 | 0.02431 | -0.62197 | -0.05707 | -0.59573 | -0.04608 | -0.65697 | good |

Modelimizi, regresyon görevlerinde yaptığımız gibi tanımlayacağız, tek bir istisna dışında. Son katmana, modelin sınıf olasılıklarını üretebilmesi için bir 'sigmoid' aktivasyonu ekleyeceğiz.

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

layers.Dense(4, activation='relu', input\_shape=[33]),

layers.Dense(4, activation='relu'),

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

])

Derleme yöntemiyle birlikte modele çapraz entropi kaybı ve doğruluk metriğini ekleyin. İki sınıflı problemler için 'ikili' versiyonları kullandığınızdan emin olun. (Daha fazla sınıf içeren problemler biraz farklı olacaktır.) Adam optimizasyon aracı sınıflandırma için de harika çalışıyor, bu yüzden onunla devam edeceğiz.

model.compile(

optimizer='adam',

loss='binary\_crossentropy',

metrics=['binary\_accuracy'],

)

Bu özel problemdeki modelin eğitiminin tamamlanması birkaç dönem alabilir, bu nedenle kolaylık olması açısından erken durdurma geri aramasını ekleyeceğiz.

early\_stopping = keras.callbacks.EarlyStopping(

patience=10,

min\_delta=0.001,

restore\_best\_weights=True,

)

history = model.fit(

X\_train, y\_train,

validation\_data=(X\_valid, y\_valid),

batch\_size=512,

epochs=1000,

callbacks=[early\_stopping],

verbose=0, *# hide the output because we have so many epochs*

)

Her zamanki gibi öğrenme eğrilerine bakacağız ve doğrulama kümesinde elde ettiğimiz kayıp ve doğruluk açısından en iyi değerleri inceleyeceğiz. (Erken durdurmanın, bu değerleri alanların ağırlıklarını geri yükleyeceğini unutmayın.)

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

*# Start the plot at epoch 5*

history\_df.loc[5:, ['loss', 'val\_loss']].plot()

history\_df.loc[5:, ['binary\_accuracy', 'val\_binary\_accuracy']].plot()

print(("Best Validation Loss: **{:0.4f}**" +\

"**\n**Best Validation Accuracy: **{:0.4f}**")\

.format(history\_df['val\_loss'].min(),

history\_df['val\_binary\_accuracy'].max()))

Best Validation Loss: 0.3534

Best Validation Accuracy: 0.8857