Introduction

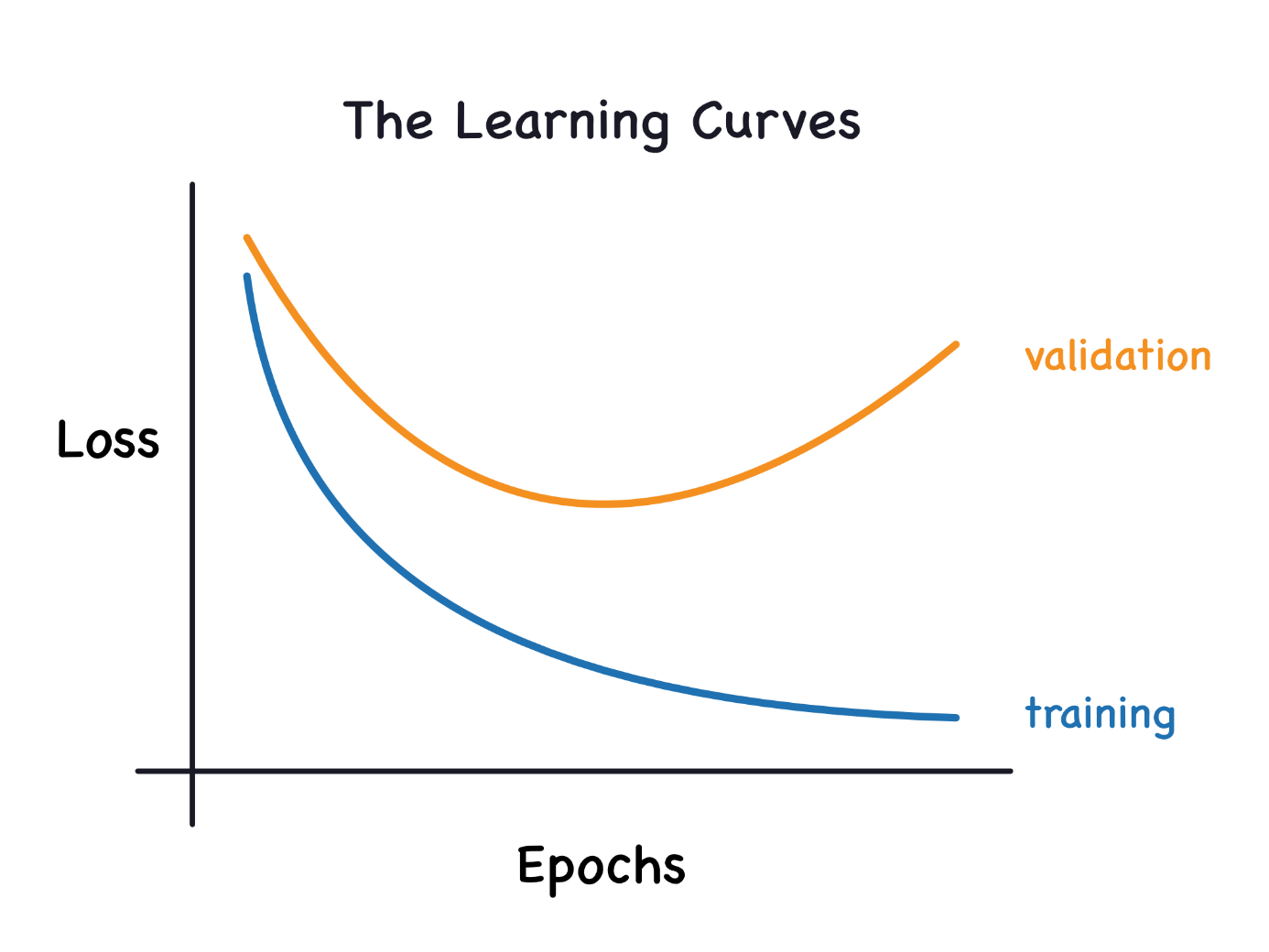
Önceki dersteki örnekten hatırlayacağınız gibi, Keras modeli eğittiği dönemler boyunca eğitim ve doğrulama kaybının geçmişini tutacaktır. Bu derste, bu öğrenme eğrilerini nasıl yorumlayacağımızı ve bunları model geliştirmeye nasıl yönlendirebileceğimizi öğreneceğiz. Özellikle, öğrenme eğrilerini yetersiz ve aşırı uyum kanıtları açısından inceleyecek ve bunu düzeltmek için birkaç stratejiye bakacağız.

# Interpreting the Learning Curves[¶](https://www.kaggle.com/code/ryanholbrook/overfitting-and-underfitting#Interpreting-the-Learning-Curves)

Eğitim verisindeki bilgiyi iki tür olarak düşünebilirsiniz: **sinyal** ve **gürültü (noise)**. Sinyal, genelleştirilebilen kısımdır, modelimizin yeni verilerden tahminler yapmasına yardımcı olabilecek kısımdır. Gürültü ise **yalnızca** eğitim verisi için geçerli olan kısımdır; gürültü, gerçek dünyadaki verilerden gelen tüm rastgele dalgalanmalar veya modelin tahminler yapmasına gerçekten yardımcı olamayan tüm tesadüfi, bilgilendirici olmayan örüntülerdir. Gürültü, yararlı gibi görünen ama aslında olmayan kısımdır.

Bir modeli, bir eğitim setindeki kaybı en aza indiren ağırlıkları veya parametreleri seçerek eğitiriz. Ancak, bir modelin performansını doğru bir şekilde değerlendirmek için onu yeni bir veri seti olan **doğrulama (validation)** verisi üzerinde değerlendirmemiz gerektiğini biliyor olabilirsiniz. (Gözden geçirme için **Makine Öğrenmesine Giriş** dersimizdeki **model doğrulama** dersine bakabilirsiniz.)

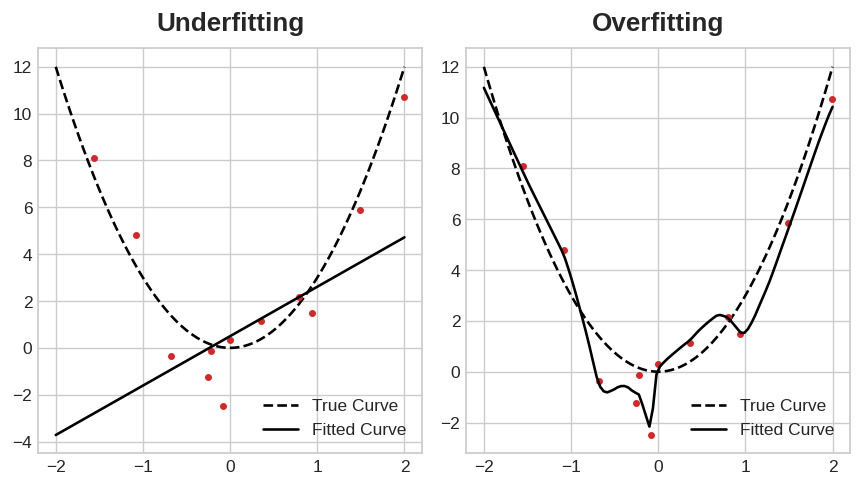
Bir modeli eğitirken, her epoch için eğitim setindeki kaybı çiziyorduk. Buna doğrulama verisinin grafiğini de ekleyeceğiz. Bu grafiklere **öğrenme eğrileri** diyoruz. Derin öğrenme modellerini etkili bir şekilde eğitmek için, bunları yorumlayabilmemiz gerekir.



Doğrulama kaybı, görülmemiş verilerdeki beklenen hatanın bir tahminini verir.

Şimdi, eğitim kaybı, model sinyali öğrendiğinde veya gürültüyü öğrendiğinde azalacaktır. Ancak doğrulama kaybı, yalnızca model sinyali öğrendiğinde azalır. (Modelin eğitim setinden öğrendiği gürültü, yeni verilere genelleşmeyecektir.) Dolayısıyla, bir model sinyali öğrendiğinde her iki eğri de aşağı iner, ancak gürültüyü öğrendiğinde eğrilerde bir **boşluk** oluşur. Boşluğun boyutu, modelin ne kadar gürültü öğrendiğini gösterir.

İdeal olarak, tüm sinyali ve hiç gürültüyü öğrenmeyen modeller oluştururduk. Bu pratik olarak asla gerçekleşmez. Bunun yerine bir takas yaparız. Daha fazla gürültü öğrenme pahasına, modelin daha fazla sinyal öğrenmesini sağlayabiliriz. Takas lehimize olduğu sürece, doğrulama kaybı düşmeye devam edecektir. Ancak belirli bir noktadan sonra takas aleyhimize dönebilir, maliyet faydayı aşar ve doğrulama kaybı yükselmeye başlar.



Bu dengeli takas, bir model eğitilirken iki sorunun ortaya çıkabileceğini gösterir: yeterli sinyal olmaması veya çok fazla gürültü olması. Eğitim setini **yetersiz uyum** (underfitting) göstermek, modelin yeterli **sinyal** öğrenemediği için kaybın olabileceğinden daha düşük olmamasıdır. Eğitim setini **aşırı uyum** (overfitting) göstermek ise, modelin çok fazla **gürültü** öğrendiği için kaybın olabileceğinden daha düşük olmamasıdır. Derin öğrenme modellerini eğitmenin püf noktası, ikisi arasındaki en iyi dengeyi bulmaktır.

Eğitim verilerinden daha fazla sinyal elde ederken gürültü miktarını azaltmanın birkaç yoluna bakacağız.

# Capacity

Bir modelin **kapasitesi**, öğrenebildiği örüntülerin boyutunu ve karmaşıklığını ifade eder. Sinir ağları için bu, büyük ölçüde sahip olduğu nöron sayısına ve bunların birbirine nasıl bağlandığına göre belirlenir. Ağınızın verileri yetersiz uyum sağladığı (underfitting) görülüyorsa, kapasitesini artırmayı denemelisiniz.

Bir ağın kapasitesini ya onu **daha geniş** (mevcut katmanlara daha fazla birim ekleyerek) ya da **daha derin** (daha fazla katman ekleyerek) yaparak artırabilirsiniz. Daha geniş ağlar, daha fazla doğrusal ilişkiyi öğrenme konusunda daha kolay bir zamana sahipken, daha derin ağlar daha doğrusal olmayanları tercih eder. Hangisinin daha iyi olduğu sadece veri setine bağlıdır.

model = keras.Sequential([

layers.Dense(16, activation='relu'),

layers.Dense(1),

])

wider = keras.Sequential([

layers.Dense(32, activation='relu'),

layers.Dense(1),

])

deeper = keras.Sequential([

layers.Dense(16, activation='relu'),

layers.Dense(16, activation='relu'),

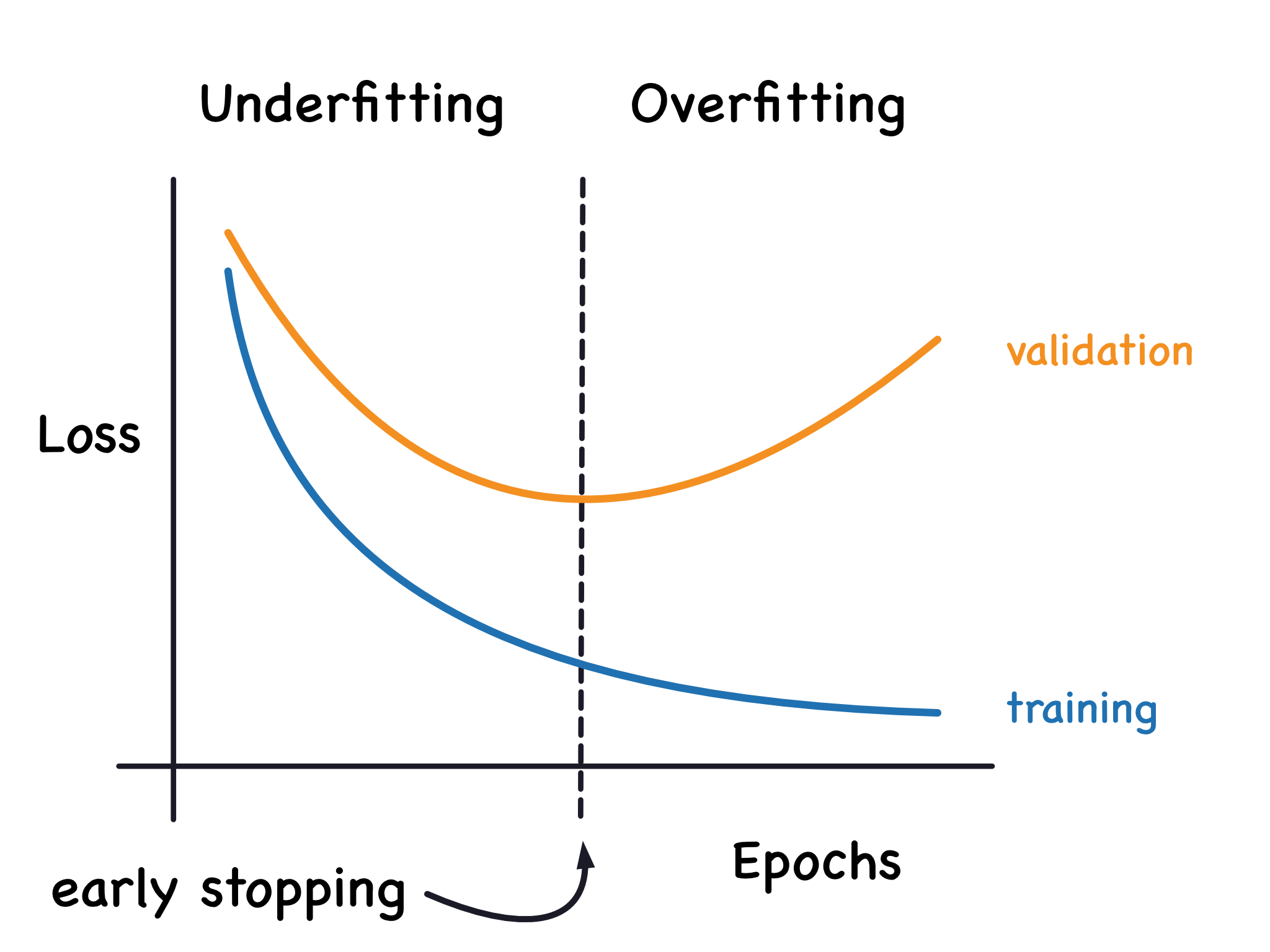
layers.Dense(1),

])

Bu alıştırmada bir ağın kapasitesinin performansını nasıl etkileyebileceğini keşfedeceksiniz.

# Early Stopping[¶](https://www.kaggle.com/code/ryanholbrook/overfitting-and-underfitting#Early-Stopping)

Bir modelin gürültüyü çok hevesle öğrendiğinde, doğrulama kaybının eğitim sırasında artmaya başlayabileceğini belirtmiştik. Bunu önlemek için, doğrulama kaybının artık düşmediği göründüğünde eğitimi durdurabiliriz. Eğitimi bu şekilde kesmeye **erken durdurma (early stopping)** denir.



Doğrulama kaybının en az olduğu modeli tutuyoruz.

Doğrulama kaybının tekrar artmaya başladığını tespit ettiğimizde, ağırlıkları minimumun meydana geldiği yere geri sıfırlayabiliriz. Bu, modelin gürültü öğrenmeye devam etmeyeceğini ve verilere aşırı uyum sağlamayacağını garanti eder.

Erken durdurma ile eğitim yapmak, ağ sinyali öğrenmeyi bitirmeden önce eğitimi çok erken durdurma tehlikesinin de daha az olduğu anlamına gelir. Dolayısıyla, çok uzun süre eğitimden kaynaklanan aşırı uyumu önlemenin yanı sıra, erken durdurma, yeterince uzun süre eğitim yapmamaktan kaynaklanan **yetersiz uyumu** da önleyebilir. Eğitim epoch'larınızı ihtiyacınız olandan daha büyük bir sayıya ayarlayın ve gerisini erken durdurma halledecektir.

## **Adding Early Stopping**[**¶**](https://www.kaggle.com/code/ryanholbrook/overfitting-and-underfitting#Adding-Early-Stopping)

Keras'ta, erken durdurmayı bir geri çağırma aracılığıyla eğitimimize dahil ediyoruz. Geri çağırma, ağ eğitilirken ara sıra çalıştırmak isteyeceğiniz bir işlevdir. Erken durdurma geri çağırması, her dönemden sonra çalışır. (Keras'ın önceden tanımlanmış çeşitli kullanışlı geri çağırmaları vardır, ancak siz de kendi geri çağırmalarınızı tanımlayabilirsiniz.)

from tensorflow.keras.callbacks import EarlyStopping

early\_stopping = EarlyStopping(

min\_delta=0.001, *# minimium amount of change to count as an improvement*

patience=20, *# how many epochs to wait before stopping*

restore\_best\_weights=True,

)

Bu parametreler şunu söylüyor: "Önceki 20 dönemde doğrulama kaybında en az 0,001'lik bir iyileşme olmadıysa, eğitimi durdurun ve bulduğunuz en iyi modeli koruyun." Doğrulama kaybının aşırı uyumdan mı yoksa rastgele toplu değişimden mi kaynaklandığını anlamak bazen zor olabilir. Parametreler, ne zaman duracağımıza dair bazı toleranslar belirlememize olanak tanır.

Örneğimizde göreceğimiz gibi, bu geri çağırmayı kayıp ve optimize ediciyle birlikte uyum yöntemine aktaracağız.

# Example - Train a Model with Early Stopping[¶](https://www.kaggle.com/code/ryanholbrook/overfitting-and-underfitting#Example---Train-a-Model-with-Early-Stopping)

Modeli, önceki eğitimdeki örnekten geliştirmeye devam edelim. Bu ağın kapasitesini artıracağız, ancak aşırı uyumu önlemek için erken durdurma geri araması da ekleyeceğiz.

İşte veri hazırlama işlemi yine burada.

import pandas as pd

from IPython.display import display

red\_wine = pd.read\_csv('../input/dl-course-data/red-wine.csv')

*# Create training and validation splits*

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

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

display(df\_train.head(4))

*# Scale to [0, 1]*

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\_)

*# Split features and target*

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

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

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

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

| fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1109 | 10.8 | 0.470 | 0.43 | 2.10 | 0.171 | 27.0 | 66.0 | 0.99820 | 3.17 | 0.76 | 10.8 | 6 |
| 1032 | 8.1 | 0.820 | 0.00 | 4.10 | 0.095 | 5.0 | 14.0 | 0.99854 | 3.36 | 0.53 | 9.6 | 5 |
| 1002 | 9.1 | 0.290 | 0.33 | 2.05 | 0.063 | 13.0 | 27.0 | 0.99516 | 3.26 | 0.84 | 11.7 | 7 |
| 487 | 10.2 | 0.645 | 0.36 | 1.80 | 0.053 | 5.0 | 14.0 | 0.99820 | 3.17 | 0.42 | 10.0 | 6 |

Şimdi ağın kapasitesini artıralım. Oldukça büyük bir ağ kullanacağız, ancak doğrulama kaybının artma belirtileri göstermesi durumunda eğitimi durdurmak için geri aramayı kullanacağız.

from tensorflow import keras

from tensorflow.keras import layers, callbacks

early\_stopping = callbacks.EarlyStopping(

min\_delta=0.001, *# minimium amount of change to count as an improvement*

patience=20, *# how many epochs to wait before stopping*

restore\_best\_weights=True,

)

model = keras.Sequential([

layers.Dense(512, activation='relu', input\_shape=[11]),

layers.Dense(512, activation='relu'),

layers.Dense(512, activation='relu'),

layers.Dense(1),

])

model.compile(

optimizer='adam',

loss='mae',

)

Geri çağırmayı tanımladıktan sonra, bunu fit'e bir argüman olarak ekleyin (birkaç tane olabilir, bu yüzden bir listeye koyun). Erken durdurmayı kullanırken, ihtiyacınız olandan daha fazla sayıda dönem seçin.

history = model.fit(

X\_train, y\_train,

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

batch\_size=256,

epochs=500,

callbacks=[early\_stopping], *# put your callbacks in a list*

verbose=0, *# turn off training log*

)

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

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

print("Minimum validation loss: **{}**".format(history\_df['val\_loss'].min()))

Minimum validation loss: 0.09269220381975174

Ve gerçekten de Keras, 500 dönemin tamamlanmasından çok önce eğitimi durdurdu!