Introduction

İlk iki derste, yoğun katman yığınlarından tam bağlantılı ağlar kurmayı öğrendik. İlk oluşturulduğunda, ağın tüm ağırlıkları rastgele ayarlanır; ağ henüz hiçbir şey "bilmez". Bu derste, bir sinir ağını nasıl eğiteceğimizi göreceğiz; yani, sinir ağlarının nasıl **öğrendiğini** göreceğiz.

Tüm makine öğrenimi görevlerinde olduğu gibi, bir eğitim veri setiyle başlarız. Eğitim verilerindeki her örnek, bazı özelliklerden (girdiler) ve beklenen bir hedefle (çıktı) oluşur. Ağı eğitmek, ağırlıklarını, özellikleri hedefe dönüştürebileceği şekilde ayarlamak demektir. Örneğin, **80 Tahıl** veri setinde, her bir tahılın **'şeker'**, **'lif'** ve **'protein'** içeriğini alıp o tahılın \*\*'kalori'\*\*si için bir tahmin üretebilecek bir ağ isteriz. Bir ağı bunu yapması için başarıyla eğitebilirsek, ağırlıkları, eğitim verilerinde ifade edilen bu özellikler ile hedef arasındaki ilişkiyi bir şekilde temsil etmelidir.

Eğitim verilerine ek olarak, iki şeye daha ihtiyacımız var:

* Ağın tahminlerinin ne kadar iyi olduğunu ölçen bir **"kayıp fonksiyonu"**.
* Ağın ağırlıklarını nasıl değiştireceğini söyleyebilen bir **"iyileştirici (optimizer)"**

# The Loss Function[¶](https://www.kaggle.com/code/ryanholbrook/stochastic-gradient-descent#The-Loss-Function)

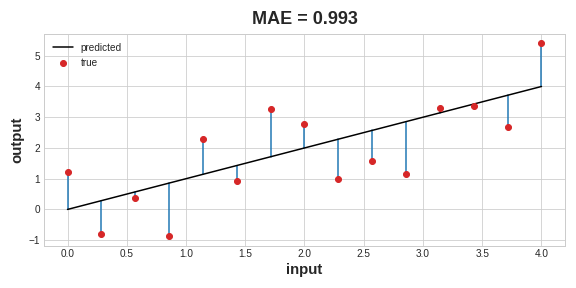
Bir ağ için mimari tasarlamayı gördük, ancak bir ağa **hangi** problemi çözeceğini nasıl söyleyeceğimizi görmedik. Bu, kayıp fonksiyonunun görevidir.

**Kayıp fonksiyonu**, hedefin gerçek değeri ile modelin tahmin ettiği değer arasındaki farkı ölçer.

Farklı problemler, farklı kayıp fonksiyonları gerektirir. Biz, görevin sayısal bir değeri tahmin etmek olduğu **regresyon** problemlerine bakıyorduk: **80 Tahıl** veri setindeki kaloriler veya **Kırmızı Şarap Kalitesi**ndeki puan. Diğer regresyon görevleri arasında bir evin fiyatını veya bir arabanın yakıt verimliliğini tahmin etmek olabilir.

Regresyon problemleri için yaygın bir kayıp fonksiyonu, **ortalama mutlak hata (mean absolute error)** veya **MAE**'dir. Her bir **y\_pred** tahmini için MAE, gerçek hedef **y\_true**'dan olan farkı, mutlak fark olan **abs(y\_true - y\_pred)** ile ölçer.

Bir veri seti üzerindeki toplam MAE kaybı, tüm bu mutlak farkların ortalamasıdır.



Ortalama mutlak hata, uydurulan eğri ile veri noktaları arasındaki ortalama uzunluktur.

MAE'nin yanı sıra, regresyon problemleri için görebileceğiniz diğer kayıp fonksiyonları **ortalama karesel hata (mean-squared error - MSE)** veya **Huber kaybıdır** (her ikisi de Keras'ta mevcuttur).

Eğitim sırasında, model ağırlıklarının doğru değerlerini bulmak için kayıp fonksiyonunu bir rehber olarak kullanır (kayıp ne kadar düşükse o kadar iyidir). Başka bir deyişle, kayıp fonksiyonu ağa hedefini söyler.

# The Optimizer - Stochastic Gradient Descent[¶](https://www.kaggle.com/code/ryanholbrook/stochastic-gradient-descent#The-Optimizer---Stochastic-Gradient-Descent)

Ağın çözmesini istediğimiz sorunu tanımladık, ancak şimdi onu **nasıl** çözeceğimizi söylememiz gerekiyor. Bu, **iyileştiricinin (optimizer)** görevidir. İyileştirici, kaybı en aza indirmek için ağırlıkları ayarlayan bir algoritmadır.

Derin öğrenmede kullanılan optimizasyon algoritmalarının neredeyse tamamı, **stokastik gradyan inişi** adı verilen bir aileye aittir. Bunlar, bir ağı adımlar halinde eğiten yinelemeli algoritmalardır. Eğitimin bir **adımı** şöyle işler:

* Bir miktar eğitim verisi örneklenir ve tahminler yapmak için ağdan geçirilir.
* Tahminler ile gerçek değerler arasındaki kayıp ölçülür.
* Son olarak, ağırlıklar kaybı azaltacak bir yönde ayarlanır.

Ardından, bu işlem kayıp istediğiniz kadar küçük olana (veya daha fazla azalmayana) kadar tekrar tekrar yapılır.



Her iterasyondaki eğitim verisi örneğine **minibatch** (veya genellikle sadece "batch") denir. Eğitim verilerinin tam bir turuna ise **epoch** adı verilir. Eğittiğiniz epoch sayısı, ağın her bir eğitim örneğini kaç kez göreceğidir.

Animasyon, 1. Dersteki doğrusal modelin SGD ile eğitildiğini gösteriyor. Soluk kırmızı noktalar tüm eğitim setini temsil ederken, koyu kırmızı noktalar minibatch'lerdir. SGD, her yeni minibatch gördüğünde, ağırlıkları (eğim olan **w** ve y-kesişimi olan **b**) o batch'teki doğru değerlerine doğru kaydırır. Batch'ten batch'e, çizgi sonunda en iyi uyumuna yakınsar. Ağırlıklar gerçek değerlerine yaklaştıkça kaybın küçüldüğünü görebilirsiniz.

## **Learning Rate and Batch Size**[**¶**](https://www.kaggle.com/code/ryanholbrook/stochastic-gradient-descent#Learning-Rate-and-Batch-Size)

Çizginin, her bir batch yönünde yalnızca küçük bir kayma yaptığına (tüm yolu gitmek yerine) dikkat edin. Bu kaymaların boyutu, **öğrenme oranı (learning rate)** tarafından belirlenir. Daha küçük bir öğrenme oranı, ağın ağırlıklarının en iyi değerlerine yakınsamadan önce daha fazla minibatch görmesi gerektiği anlamına gelir.

Öğrenme oranı ve minibatch'lerin boyutu, SGD eğitiminin nasıl ilerleyeceği üzerinde en büyük etkiye sahip iki parametredir. Etkileşimleri genellikle inceliklidir ve bu parametreler için doğru seçimi yapmak her zaman açık değildir. (Bu etkileri alıştırmada keşfedeceğiz.)

Neyse ki, çoğu çalışma için tatmin edici sonuçlar elde etmek için kapsamlı bir hiperparametre araması yapmak gerekli olmayacaktır. **Adam**, uyarlanabilir bir öğrenme oranına sahip bir SGD algoritmasıdır ve bu, herhangi bir parametre ayarlaması gerektirmeden çoğu problem için uygun olmasını sağlar (bir anlamda "kendi kendine ayarlanır"). Adam, harika bir genel amaçlı iyileştiricidir.

## **Adding the Loss and Optimizer**[**¶**](https://www.kaggle.com/code/ryanholbrook/stochastic-gradient-descent#Adding-the-Loss-and-Optimizer)

Bir modeli tanımladıktan sonra, modelin derleme yöntemiyle bir kayıp fonksiyonu ve iyileştirici ekleyebilirsiniz:

model.compile(

optimizer="adam",

loss="mae",

)

Kaybı ve optimize ediciyi yalnızca bir dizeyle belirleyebildiğimizi unutmayın. Bunlara doğrudan Keras API'si üzerinden de erişebilirsiniz (örneğin, parametreleri ayarlamak isterseniz), ancak bizim için varsayılanlar sorunsuz çalışacaktır.

İsimde Ne Var?

Eğim, ağırlıkların hangi yöne gitmesi gerektiğini söyleyen bir vektördür. Daha doğrusu, kaybın en hızlı şekilde değişmesi için ağırlıkları nasıl değiştireceğimizi söyler. Sürecimize gradyan inişi adını veriyoruz çünkü kayıp eğrisini minimuma doğru indirmek için eğimi kullanıyor. Stokastik, "şans eseri belirlenmiş" anlamına gelir. Eğitimimiz stokastiktir çünkü mini gruplar veri kümesinden rastgele örneklerdir. İşte bu yüzden adı SGD!

# Example - Red Wine Quality[¶](https://www.kaggle.com/code/ryanholbrook/stochastic-gradient-descent#Example---Red-Wine-Quality)

Artık derin öğrenme modellerini eğitmeye başlamak için ihtiyacımız olan her şeyi biliyoruz. Öyleyse bunu uygulamada görelim! Kırmızı Şarap Kalitesi veri setini kullanacağız.

Bu veri seti, yaklaşık 1600 Portekiz kırmızı şarabından alınan fizikokimyasal ölçümlerden oluşmaktadır. Ayrıca, her şarap için kör tadım testlerinden alınan bir kalite derecelendirmesi de bulunmaktadır. Bu ölçümlerden bir şarabın algılanan kalitesini ne kadar iyi tahmin edebiliriz?

Tüm veri hazırlama işlemlerini bir sonraki gizli hücreye yerleştirdik. Bu, bundan sonraki kısım için gerekli olmadığından, bu kısmı atlayabilirsiniz. Ancak şimdilik dikkat etmeniz gereken bir nokta, her bir özelliği [0,1] aralığında yer alacak şekilde yeniden ölçeklendirmiş olmamızdır. 5. Ders'te daha detaylı olarak tartışacağımız gibi, sinir ağları, girdileri ortak bir ölçekte olduğunda en iyi performansı gösterme eğilimindedir.

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 |

Bu ağda kaç girdi olmalı? Bunu, veri matrisindeki sütun sayısına bakarak öğrenebiliriz. Hedefi ('kalite') buraya eklemediğinizden emin olun; yalnızca girdi özelliklerini ekleyin.

print(X\_train.shape)

(1119, 11)

On bir sütun, on bir girdi anlamına gelir.

1500'den fazla nörona sahip üç katmanlı bir ağ seçtik. Bu ağ, verilerdeki oldukça karmaşık ilişkileri öğrenebilmelidir.

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

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

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

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

layers.Dense(1),

])

Modelinizin mimarisine karar vermek bir sürecin parçası olmalıdır. Basit bir şekilde başlayın ve doğrulama kaybını rehberiniz olarak kullanın. Alıştırmalarda model geliştirme hakkında daha fazla bilgi edineceksiniz.

Modeli tanımladıktan sonra, optimizasyon ve kayıp fonksiyonunu derliyoruz.

model.compile(

optimizer='adam',

loss='mae',

)

Artık eğitime başlamaya hazırız! Keras'a, optimize ediciye eğitim verilerinden 256 satırı (batch\_size) tek seferde beslemesini ve bunu veri kümesi boyunca 10 kez yapmasını (epochs) söyledik.

history = model.fit(

X\_train, y\_train,

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

batch\_size=256,

epochs=10,

)

Epoch 1/10

5/5 [==============================] - 1s 66ms/step - loss: 0.2470 - val\_loss: 0.1357

Epoch 2/10

5/5 [==============================] - 0s 21ms/step - loss: 0.1349 - val\_loss: 0.1231

Epoch 3/10

5/5 [==============================] - 0s 23ms/step - loss: 0.1181 - val\_loss: 0.1173

Epoch 4/10

5/5 [==============================] - 0s 21ms/step - loss: 0.1117 - val\_loss: 0.1066

Epoch 5/10

5/5 [==============================] - 0s 22ms/step - loss: 0.1071 - val\_loss: 0.1028

Epoch 6/10

5/5 [==============================] - 0s 20ms/step - loss: 0.1049 - val\_loss: 0.1050

Epoch 7/10

5/5 [==============================] - 0s 20ms/step - loss: 0.1035 - val\_loss: 0.1009

Epoch 8/10

5/5 [==============================] - 0s 20ms/step - loss: 0.1019 - val\_loss: 0.1043

Epoch 9/10

5/5 [==============================] - 0s 19ms/step - loss: 0.1005 - val\_loss: 0.1035

Epoch 10/10

5/5 [==============================] - 0s 20ms/step - loss: 0.1011 - val\_loss: 0.0977

Keras'ın, model eğitilirken sizi kayıp konusunda güncel tutacağını görebilirsiniz.

Ancak genellikle kaybı görüntülemenin daha iyi bir yolu, onu çizmektir. Fit yöntemi, eğitim sırasında oluşan kaybın kaydını bir Geçmiş nesnesinde tutar. Verileri, çizim işlemini kolaylaştıran bir Pandas veri çerçevesine dönüştüreceğiz.

import pandas as pd

*# convert the training history to a dataframe*

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

*# use Pandas native plot method*

history\_df['loss'].plot();

Kayıpların dönemler geçtikçe nasıl azaldığına dikkat edin. Kayıp eğrisi bu şekilde yatay hale geldiğinde, modelin öğrenebileceği her şeyi öğrendiği ve ek dönemler için devam etmenin bir anlamı olmadığı anlamına gelir.