Welcome to Deep Learning![¶](https://www.kaggle.com/code/ryanholbrook/a-single-neuron#Welcome-to-Deep-Learning!)

Kaggle'ın Derin Öğrenmeye Giriş kursuna hoş geldiniz! Kendi derin sinir ağlarınızı kurmaya başlamak için ihtiyacınız olan her şeyi öğrenmek üzeresiniz. Keras ve Tensorflow kullanarak şunları nasıl yapacağınızı öğreneceksiniz:

* Tam bağlantılı (fully-connected) bir sinir ağı mimarisi oluşturmak
* Sinir ağlarını iki klasik makine öğrenmesi problemine uygulamak: **regresyon** ve **sınıflandırma**
* Sinir ağlarını **stokastik gradyan inişi** ile eğitmek
* **Dropout**, **toplu normalleştirme (batch normalization)** ve diğer tekniklerle performansı artırmak

Eğitimler, bu konuları tam olarak çalışılmış örneklerle size tanıtacak, alıştırmalarda ise bu konuları daha derinlemesine keşfedecek ve gerçek dünya veri kümelerine uygulayacaksınız.

Hadi başlayalım!

# What is Deep Learning?[¶](https://www.kaggle.com/code/ryanholbrook/a-single-neuron#What-is-Deep-Learning?)

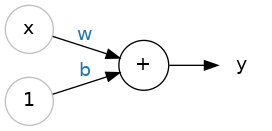
Yapay zekâ alanındaki son yılların en etkileyici gelişmelerinden bazıları **derin öğrenme** alanında gerçekleşmiştir. Doğal dil çevirisi, görüntü tanıma ve oyun oynama gibi görevlerin tümü, derin öğrenme modellerinin insan seviyesindeki performansa yaklaştığı veya onu aştığı alanlardır.

Peki, derin öğrenme nedir? **Derin öğrenme**, derin hesaplama katmanlarıyla karakterize edilen bir makine öğrenimi yaklaşımıdır. Bu hesaplama derinliği, derin öğrenme modellerinin en zorlu gerçek dünya veri kümelerinde bulunan karmaşık ve hiyerarşik örüntüleri çözmesini sağlamıştır.

Güçleri ve ölçeklenebilirlikleri sayesinde **sinir ağları**, derin öğrenmenin en belirgin modeli haline gelmiştir. Sinir ağları, her bir nöronun yalnızca basit bir hesaplama yaptığı nöronlardan oluşur. Bir sinir ağının gücü ise bu nöronların oluşturabildiği bağlantıların karmaşıklığından gelir.

# The Linear Unit[¶](https://www.kaggle.com/code/ryanholbrook/a-single-neuron" \l "The-Linear-Unit" \t "_self)

Öyleyse, bir sinir ağının temel bileşeni olan bireysel nöronla başlayalım. Bir diyagram olarak, tek girdili bir nöron (veya birim) şöyle görünür:

The Linear Unit:y=wx+b

Girdi **x**'tir. Nörona olan bağlantısının bir **ağırlığı (weight)** vardır, o da **w**'dir. Bir değer bir bağlantıdan aktığında, o değeri bağlantının ağırlığı ile çarparsınız. **x** girdisi için nörona ulaşan değer **w \* x**'tir. Bir sinir ağı, ağırlıklarını değiştirerek "öğrenir".

**b** dediğimiz şey, **önyargı (bias)** olarak adlandırdığımız özel bir ağırlık türüdür. Önyargının ilişkili olduğu herhangi bir girdi verisi yoktur; bunun yerine, diyagrama bir **1** koyarız, böylece nörona ulaşan değer sadece **b** olur (1∗b=b olduğu için). Önyargı, nöronun çıktıyı girdilerinden bağımsız olarak değiştirmesini sağlar.

**y**, nöronun nihayetinde ürettiği değerdir. Çıktıyı elde etmek için, nöron bağlantıları aracılığıyla aldığı tüm değerleri toplar. Bu nöronun aktivasyonu y=w⋅x+b'dir veya bir formül olarak y=w⋅x+b şeklinde ifade edilir.

y=wx+b formülü tanıdık geliyor mu?

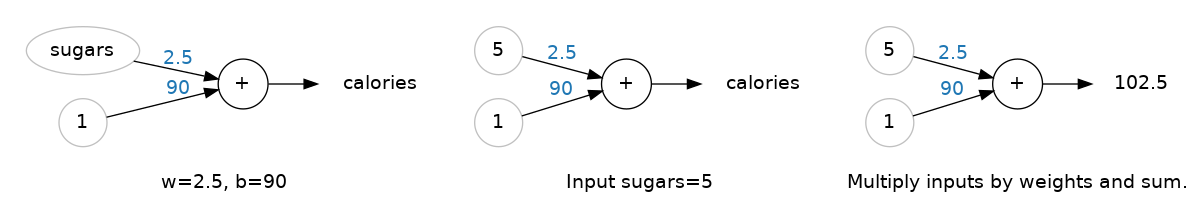
Bu bir doğru denklemi! Eğim-kesişim denklemi, burada w eğim ve b y-kesişimidir.

# Example - The Linear Unit as a Model[¶](https://www.kaggle.com/code/ryanholbrook/a-single-neuron#Example---The-Linear-Unit-as-a-Model)

## Tek Bir Nöron Modeliyle Başlangıç

Tekil nöronlar genellikle daha büyük bir ağın parçası olarak işlev görse de, bir başlangıç noktası olarak tek bir nöron modeliyle başlamak faydalıdır. Tekil nöron modelleri **doğrusal** modellerdir.

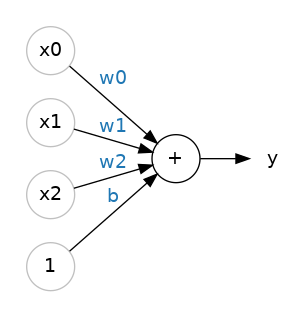
Bunun, **80 Tahıl (80 Cereals)** gibi bir veri kümesinde nasıl çalışabileceğini düşünelim. Girdi olarak **'şeker'** (servis başına gram cinsinden şeker) ve çıktı olarak **'kalori'** (servis başına kalori) ile bir modeli eğitirsek, önyargının **b=90** ve ağırlığın **w=2.5** olduğunu bulabiliriz. Servis başına 5 gram şeker içeren bir tahılın kalori içeriğini şu şekilde tahmin edebiliriz:



Formülümüze göre kontrol ettiğimizde, beklediğimiz gibi kalori calories=2.5×5+90=102.5

# Multiple Inputs[¶](https://www.kaggle.com/code/ryanholbrook/a-single-neuron" \l "Multiple-Inputs" \t "_self)

**80 Tahıl** veri kümesi, yalnızca \*\*'şeker'\*\*den çok daha fazla özelliğe sahiptir. Modelimizi lif veya protein içeriği gibi şeyleri içerecek şekilde genişletmek istersek ne olur? Bu oldukça kolaydır. Nörona, her ek özellik için bir tane olmak üzere daha fazla girdi bağlantısı ekleyebiliriz. Çıktıyı bulmak için, her bir girdiyi bağlantı ağırlığıyla çarpar ve sonra hepsini bir araya toplarız.

Üç girişli doğrusal bir birim.

Bu nöronun formülü şöyle olacaktır:

y=w0x0+w1x1+w2x2+b

İki girdisi olan doğrusal bir birim bir düzleme, bundan daha fazla girdisi olan bir birim ise bir hiperdüzleme uyacaktır.

# Linear Units in Keras[¶](https://www.kaggle.com/code/ryanholbrook/a-single-neuron" \l "Linear-Units-in-Keras" \t "_self)

Keras'ta bir model oluşturmanın en kolay yolu, bir sinir ağını katmanlar yığını olarak oluşturan **keras.Sequential**'dır. Yukarıdakiler gibi modelleri **yoğun katman (dense layer)** kullanarak oluşturabiliriz (bu konu hakkında bir sonraki derste daha fazla bilgi edineceğiz).

Üç girdi özelliği ('şeker', 'lif' ve 'protein') kabul eden ve tek bir çıktı ('kalori') üreten doğrusal bir modeli şu şekilde tanımlayabiliriz:

from tensorflow import keras

from tensorflow.keras import layers

*# Create a network with 1 linear unit*

model = keras.Sequential([

layers.Dense(units=1, input\_shape=[3])

])

İlk argüman olan **units** ile kaç tane çıktı istediğimizi tanımlarız. Bu durumda sadece \*\*'kalori'\*\*yi tahmin ettiğimiz için **units=1** kullanacağız.

İkinci argüman olan **input\_shape** ile Keras'a girdilerin boyutlarını söyleriz. **input\_shape=[3]** olarak ayarlamak, modelin üç özelliği girdi olarak kabul etmesini sağlar ('şeker', 'lif' ve 'protein').

Bu model artık eğitim verilerine uydurulmaya hazırdır!

input\_shape neden bir Python listesidir?

Bu kursta kullanacağımız veriler, Pandas veri çerçevesindeki gibi tablo verileri olacaktır. Veri kümesindeki her özellik için bir girdimiz olacak. Özellikler sütunlara göre düzenlendiğinden, her zaman input\_shape=[sütun\_sayısı] değerini kullanacağız. Keras'ın burada bir liste kullanmasının nedeni, daha karmaşık veri kümelerinin kullanımına olanak sağlamaktır. Örneğin, görüntü verilerinin üç boyuta ihtiyacı olabilir: [yükseklik, genişlik, kanallar].