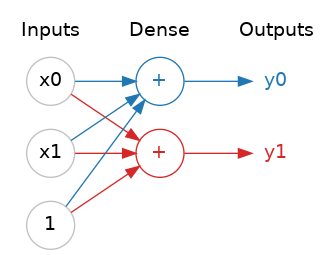
Introduction

Bu derste, derin sinir ağlarının meşhur olduğu karmaşık ilişki türlerini öğrenebilen sinir ağlarını nasıl kurabileceğimizi göreceğiz.

Buradaki anahtar fikir **modülerlik**; yani, daha basit işlevsel birimlerden karmaşık bir ağ oluşturmaktır. Doğrusal bir birimin nasıl doğrusal bir fonksiyon hesapladığını gördük, şimdi ise daha karmaşık ilişkileri modellemek için bu tekil birimleri nasıl birleştirip değiştireceğimizi göreceğiz.

# Layers[¶](https://www.kaggle.com/code/ryanholbrook/deep-neural-networks" \l "Layers" \t "_self)

Sinir ağları, nöronlarını genellikle **katmanlar** halinde düzenler. Ortak bir girdi setine sahip doğrusal birimleri bir araya getirdiğimizde, bir **yoğun (dense)** katman elde ederiz.



İki girdi ve bir önyargı alan iki doğrusal birimden oluşan yoğun bir katman.

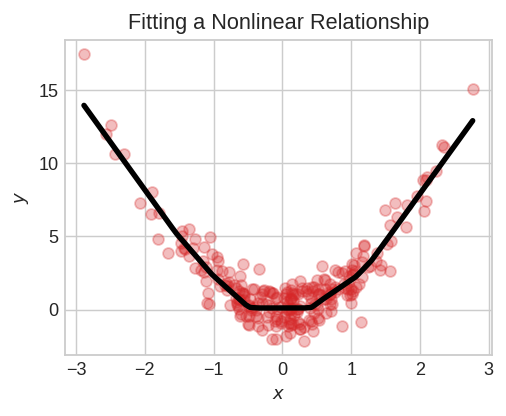
Bir sinir ağındaki her katmanın, bir tür göreceli olarak basit dönüşüm gerçekleştirdiğini düşünebilirsiniz. Derin bir katman yığını aracılığıyla, bir sinir ağı girdilerini giderek daha karmaşık yollarla dönüştürebilir. İyi eğitilmiş bir sinir ağında, her katman bizi çözüme biraz daha yaklaştıran bir dönüşümdür.

## Birçok Katman Çeşidi

Keras'ta bir "katman" çok genel bir şeydir. Bir katman, esasen, herhangi bir tür **veri dönüşümü** olabilir. **Evrişimsel (convolutional)** ve **tekrarlayan (recurrent)** katmanlar gibi birçok katman, nöronları kullanarak veriyi dönüştürür ve esas olarak oluşturdukları bağlantı düzeninde farklılık gösterir. Ancak diğerleri, **özellik mühendisliği** veya sadece **basit aritmetik** için kullanılır. Keşfedilecek koca bir katman dünyası var; bir göz atın!

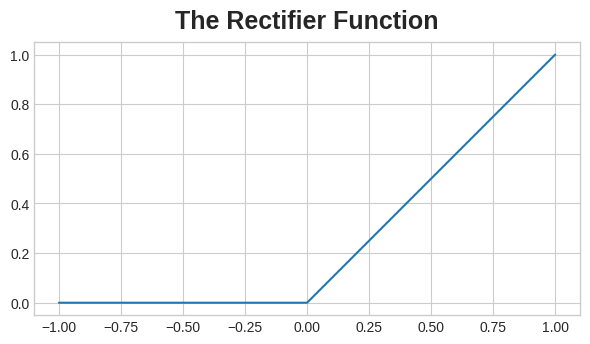
# The Activation Function[¶](https://www.kaggle.com/code/ryanholbrook/deep-neural-networks" \l "The-Activation-Function" \t "_self)

Ancak, arada hiçbir şey olmadan iki yoğun katmanın, tek bir yoğun katmandan daha iyi olmadığı ortaya çıkmıştır. Yoğun katmanlar kendi başlarına bizi çizgiler ve düzlemler dünyasının dışına asla çıkaramaz. İhtiyacımız olan şey **doğrusal olmayan** bir şeydir. İhtiyacımız olan şey **aktivasyon fonksiyonları**dır.



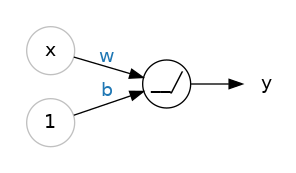
Aktivasyon fonksiyonları olmadan, sinir ağları yalnızca doğrusal ilişkileri öğrenebilir. Eğrileri uydurmak için aktivasyon fonksiyonlarını kullanmamız gerekir.

Bir aktivasyon fonksiyonu, bir katmanın her bir çıktısına (aktivasyonlarına) uyguladığımız bir fonksiyondur. En yaygın olanı, doğrultucu fonksiyonudur max(0,x)



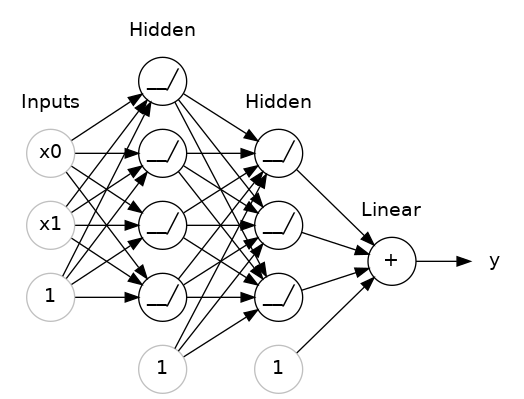
Doğrultucu (rectifier) fonksiyonu, negatif kısmının "doğrultularak" sıfıra çekildiği bir çizgi grafiğine sahiptir. Bu fonksiyonu bir nöronun çıktılarına uygulamak, veriye bir **kıvrım** katacak ve bizi basit çizgilerden uzaklaştıracaktır.

Doğrultucuyu doğrusal bir birime eklediğimizde, **doğrultulmuş doğrusal birim (rectified linear unit)** veya **ReLU** elde ederiz. (Bu nedenle, doğrultucu fonksiyonuna "ReLU fonksiyonu" demek yaygındır.) Doğrusal bir birime ReLU aktivasyonu uygulamak, çıktının **max(0, w \* x + b)** olacağı anlamına gelir ve bunu bir diyagramda şöyle çizebiliriz:

 A rectified linear unit

# Stacking Dense Layers[¶](https://www.kaggle.com/code/ryanholbrook/deep-neural-networks" \l "Stacking-Dense-Layers" \t "_self)

Artık biraz doğrusal olmayanlığa sahip olduğumuza göre, karmaşık veri dönüşümleri elde etmek için katmanları nasıl istifleyebileceğimizi görelim.



Yoğun katmanların bir araya gelmesiyle "tam bağlantılı" bir ağ oluşur.

Çıktı katmanından önceki katmanlara bazen **gizli (hidden)** katmanlar denir, çünkü çıktılarını doğrudan göremeyiz.

Şimdi, son (çıktı) katmanının doğrusal bir birim (yani, aktivasyon fonksiyonu olmayan) olduğuna dikkat edin. Bu durum, bu ağı, keyfi bir sayısal değeri tahmin etmeye çalıştığımız regresyon görevi için uygun hale getirir. Diğer görevler (sınıflandırma gibi) çıktıda bir aktivasyon fonksiyonu gerektirebilir.

## **Building Sequential Models[¶](https://www.kaggle.com/code/ryanholbrook/deep-neural-networks" \l "Building-Sequential-Models" \t "_self)**

Kullandığımız Sıralı model, bir katman listesini ilk katmandan son katmana doğru birbirine bağlayacaktır: ilk katman girdiyi alır, son katman çıktıyı üretir. Bu, yukarıdaki şekildeki modeli oluşturur:

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

*# the hidden ReLU layers*

layers.Dense(units=4, activation='relu', input\_shape=[2]),

layers.Dense(units=3, activation='relu'),

*# the linear output layer*

layers.Dense(units=1),

])

Tüm katmanları ayrı argümanlar olarak değil, [katman, katman, katman, ...] şeklinde bir liste halinde birlikte ilettiğinizden emin olun. Bir katmana aktivasyon fonksiyonu eklemek için, aktivasyon argümanına adını girmeniz yeterlidir.