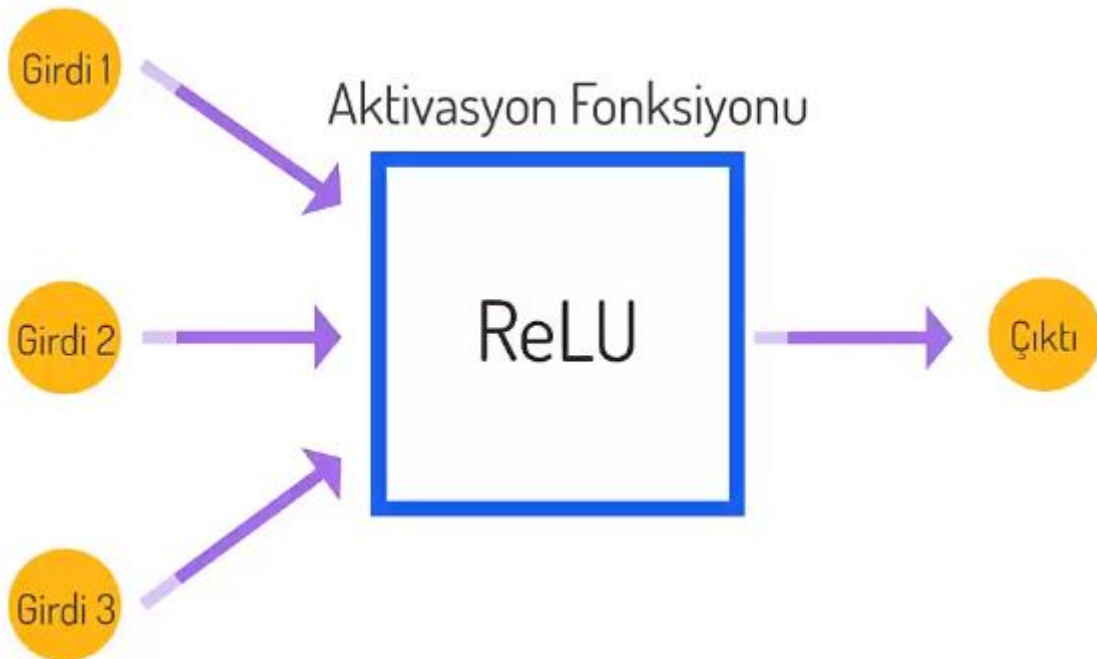
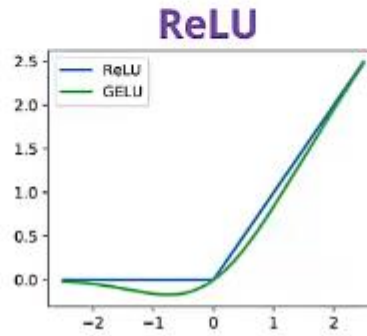
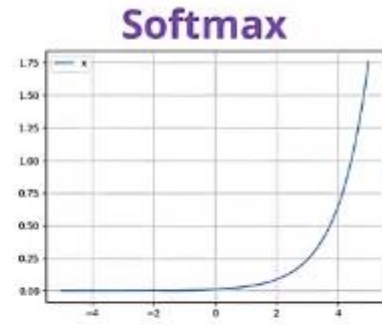
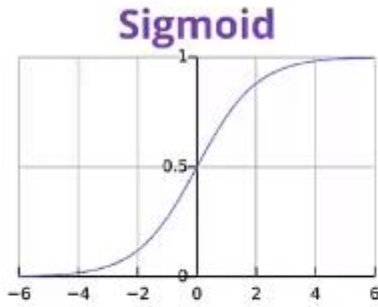
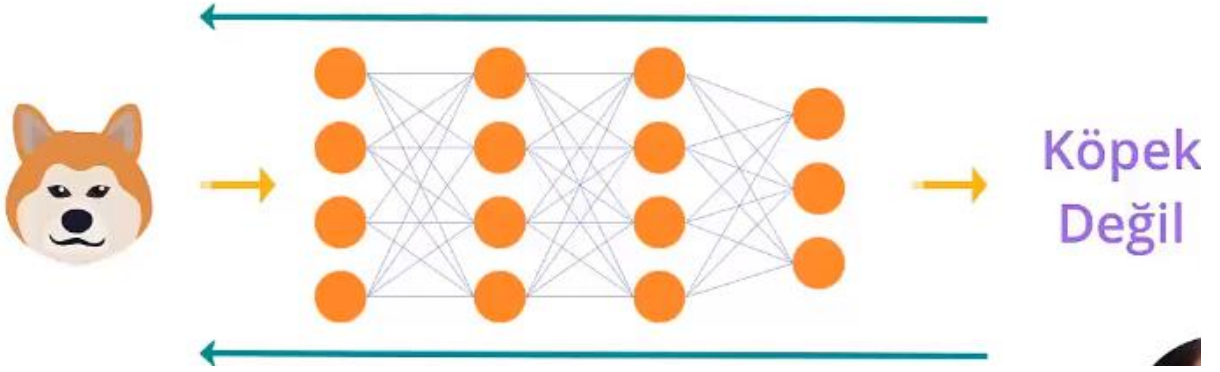


# YAPAY SİNİR AĞLARI

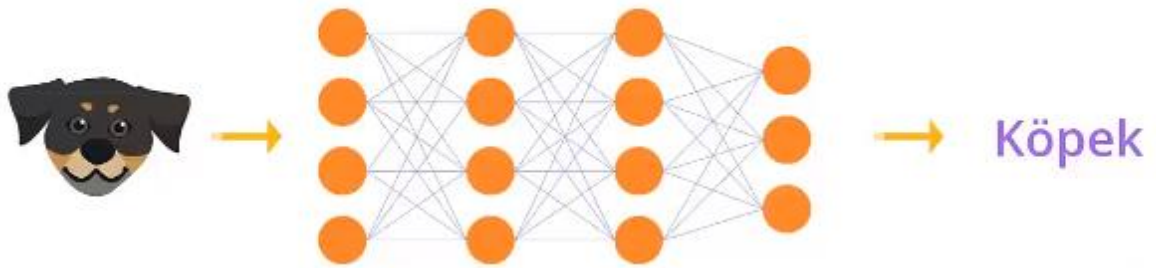
Verilerin ya da hesaplamaların tamamı dikkate alınmaz. Sadece belirli eşik değerin üzerinde kalanlar dikkate alınarak hesaplamaya dahil edilir. Bu eşik değeri hesaplamak için de çeşitli fonksiyonlar kullanırız. Sigmoid, ReLU, Softmax

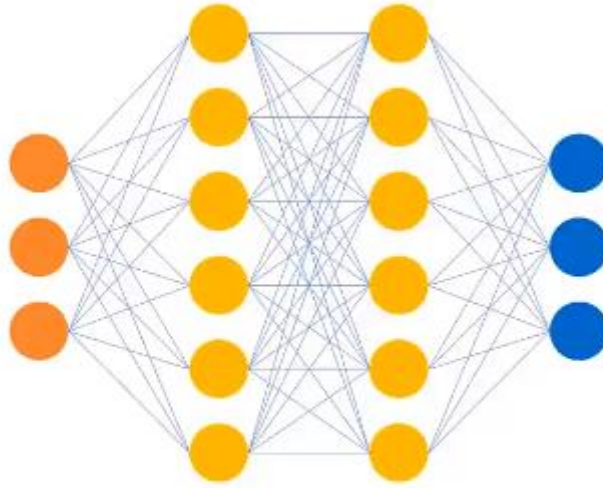


Bu diyagramda gördüğümüz gibi input dediğimiz girdiler aktivasyon fonksiyonuna dahil ediliyor. Aktivasyon fonksiyonundan geçebilenler output olarak yani çıktı olarak bize veriliyor. ReLU aktivasyon fonksiyonu, eğer girdi değeri pozitif ise dikkate alır. Negatif ise o zaman 0 sonucunu verir.

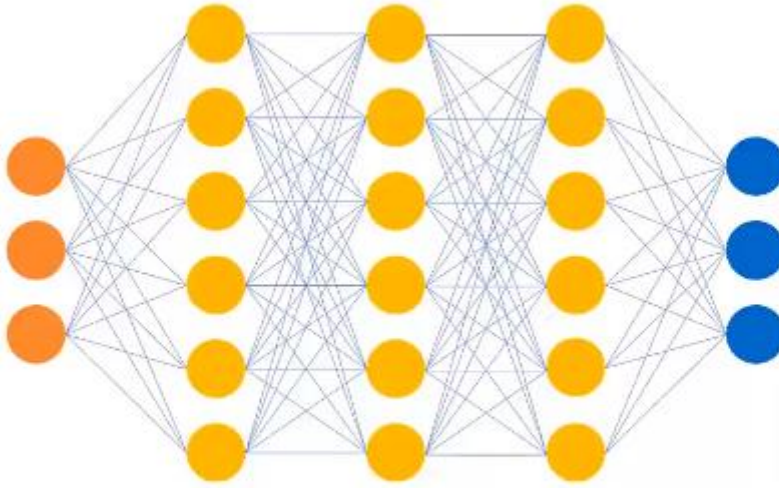


Doğru çıktı sonucu alamadığımızda yani hata meydana geldiğinde yitim hesaplaması dediğimiz bir hesaplama yöntemiyle yapılan hata oranı ölçülerek, geriye doğru yayılım yapılır. Yapılan bu hatayı daha önce hesaplama yapmış olan nöronlara bildirilir. Nöronlar tekrar hesaplama yaparak daha doğru sonuçlarla tekrar dönüş sağlarlar.





● Girdi katmanı ● Gizli katman ● Çıktı katmanı



● Girdi katmanı ● Gizli katman ● Çıktı katmanı

## Yapay Sinir Ağları Uygulama

Kütüphaneler:



```
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
```

Katmanları tanımlamak için Dense kullanıldı. Seaborn kütüphanesinden iris verisini çekiyoruz.

İris datasını yükleme:

```
iris = sns.load_dataset('iris')
iris
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows x 5 columns

Çiçek isimlerini bu şekilde kullanamayacağımız için encode etmeliyiz. Yani 0, 1, 2, 3 diye kaç tane sınıf varsa bunlara göre numaralandıracağız. OneHotEncoder ile bunu yapacağız. Bizim 3 tane olduğu için 0, 1, 2 olacak.

```
[3] x = iris.drop(columns=['species'])
     y = iris['species'].values

[4] enc = OneHotEncoder()
     y = enc.fit_transform(y[:, np.newaxis]).toarray()


x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=42)
```

Modelimizin mimarisini oluşturma: katmanlardaki önemli olan giriş ve çıkış değerleri önemli, doğru değerler girilmelidir. 4 tane nörona ihtiyacımız var girişte. Çıkışta da 3 çeşit çiçeğimiz olduğu için 3 nörona ihtiyacımız var. Yetim hesaplaması için binary\_crossentropy ile hesapla. binary\_crossentropy, her bir sonucun gerçek sonuçtan olan farkına bak. Metrics'de accuracy (doğruluk) olsun. Yani bulduğun model ne kadar doğruya yakın ise o kadar iyidir. Epoch, tekrar sayısı.

```
[7] model = Sequential()
```

```
[8] model.add(Dense(8, input_dim = x_train.shape[1],activation = 'relu'))  
    model.add(Dense(6,activation='sigmoid'))  
    model.add(Dense(8, activation='relu'))  
    model.add(Dense(y_train.shape[1],activation='softmax'))
```

```
[9] model.compile(loss='binary_crossentropy',metrics=['accuracy'])
```

```
 model.fit(x_train,y_train, epochs=150)
```

```
Epoch 1/150  
4/4 [=====] - 1s 3ms/step - loss: 0.6580 - accuracy: 0.3632  
Epoch 2/150  
4/4 [=====] - 0s 3ms/step - loss: 0.6524 - accuracy: 0.3570  
Epoch 3/150  
4/4 [=====] - 0s 3ms/step - loss: 0.6477 - accuracy: 0.3737  
Epoch 4/150  
4/4 [=====] - 0s 4ms/step - loss: 0.6472 - accuracy: 0.3278  
Epoch 5/150
```

•

•

•

```
4/4 [=====] - 0s 5ms/step - loss: 0.3062 - accuracy: 0.9586  
Epoch 145/150  
4/4 [=====] - 0s 5ms/step - loss: 0.2996 - accuracy: 0.9484  
Epoch 146/150  
4/4 [=====] - 0s 3ms/step - loss: 0.2961 - accuracy: 0.9701  
Epoch 147/150  
4/4 [=====] - 0s 3ms/step - loss: 0.2984 - accuracy: 0.9494  
Epoch 148/150  
4/4 [=====] - 0s 4ms/step - loss: 0.3066 - accuracy: 0.9577  
Epoch 149/150  
4/4 [=====] - 0s 6ms/step - loss: 0.2958 - accuracy: 0.9436  
Epoch 150/150  
4/4 [=====] - 0s 3ms/step - loss: 0.2914 - accuracy: 0.9577  
<tensorflow.python.keras.callbacks.History at 0x7f299926c390>
```

Sınıfları tahmin etme:



```
model.predict_classes(x_test)
```

```
array([1, 0, 2, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 1,  
       0, 2, 2, 1, 2, 2, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0])
```

## KAYNAKÇA

Bilgeiř “Herkes iin Yapay Zekâ II” eđitimi.

**KODLUYORUZ**  
geleceđi kodluyoruz >\_

 **EMpower**  
Enriching young lives in emerging markets