Derin Öğrenme

Yazılım Mühendisliği Bölümü
Atılım Üniversitesi

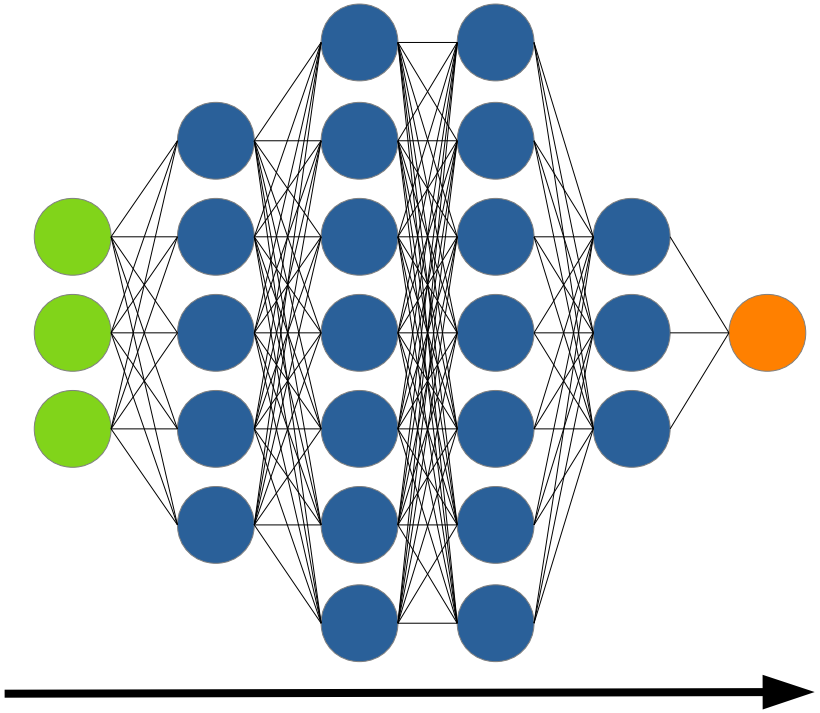
Tolga Üstünkök

Ön Bilgilendirmeler

- Bazı İngilizce terimler oldukları gibi bırakılmışlardır. Bu terimleri slaytlar içerisinde *italic* yazılmış olarak görebilirsiniz.
- Makine öğrenmesi yöntemlerinin isimleri Türkçe'ye çevrilmeyip oldukları gibi bırakılmışlardır.
- Yapay sinir ağları, *computer vision*, zaman serileri hakkında az da olsa ön bilgi gereklidir.

Vanilla Deep Neural Networks

- Vanilla Deep Neural Network'ler (DNN) aslında birden fazla saklı katmanı[*hidden layer*] olan sıradan ağlardır.
- Buradaki önemli nokta eklenen her katman ağın öğrenme potansiyelini ciddi ölçüde artırır.
- Ancak öğrenme işlemi ise git gide zorlaşır ve öğrenmenin yeterince iyi olması için elinizde daha çok veri olması gerekir.



Vanilla Deep Neural Networks

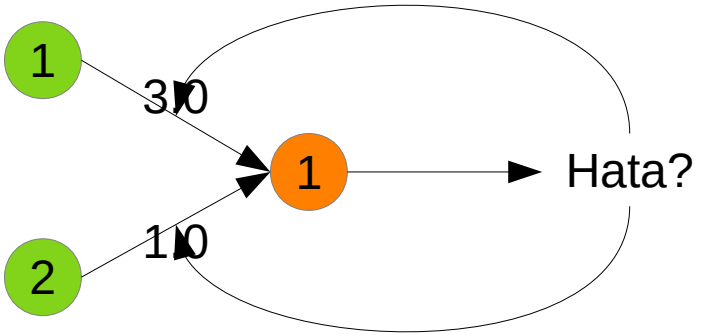
- Bir ağda kaç tane katman kullanmanız gerektiğiyle ilgili bir kural ya da kısıtlama yoktur.
- Ancak son araştırmalarda bu konuyla ilgili bazı öneriler mevcuttur.
- Şimdi bir de öğrenmenin nasıl yapıldığıyla ilgili detaylara bir göz atalım.

Backpropagation (Linnainmaa, S. 1970)

- Backpropagation (BP) ilk defa 1970 yılımda Fin bir master öğrencisi olan Seppo Linnainmaa tarafından geliştirilmiştir.
- Ancak BP'nin esas meşhur oluşu ünlü "*Learning representations by back-propagating errors*" Rumelhart, D. E., Hinton, G., Williams R. 1986 makalesi sayesindedir.
- Merak edenler için:
  - Learning representations by back-propagating errors

Backpropagation

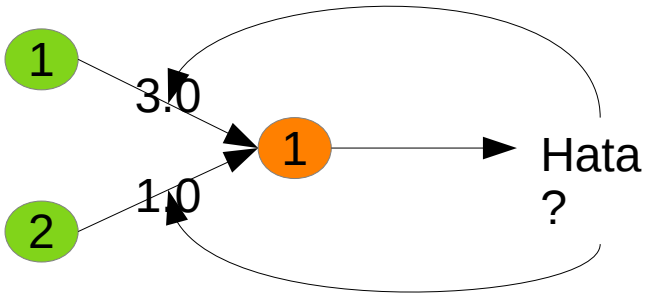
- Öncelikle kendimize oldukça basit bir ağ hazırlayalım.



- Soru şu: üretilen çıktıdaki hatadan hangi ağırlık ne kadar sorumlu? Her ikisi de yarı yarıya mı? Yoksa daha mı farklı?

Backpropagation

- Aslında cevap oldukça bariz. Hata bütün yollardan ağırlıklarıyla orantılı şekilde dağıtılır.
- Önceki ağı hatırlayalım:
  - Üstteki yolun hata üzerindeki payı ¾,
  - Alttaki yolun hata üzerindeki payı da ¼ olacaktır.



- Bunun sebebi, üstteki yoldan gelen girdideki değişim, ağırlığın fazla olmasından dolayı hatayı daha hızlı bir şekilde, aşağıdan gelen girdi de ağırlığın az olmasından dolayı hatayı daha yavaş bir şekilde değiştirmesidir.

Backpropagation

- Bu fikir kaç katman olursa olsun ağın çıktısında hesaplanan hatadan girdi katmanına kadar bütün katmanlardaki nöronlar için tek tek hesaplanır.
- Her katmandaki nöronların çıktılarındaki hataları hesapladık. Peki bunları kullanarak ağırlıkları nasıl değiştireceğiz?

Backpropagation

- İşte burada Calculus’ten biraz destek almamız gerekiyor.
- Çünkü işin içine birşeyin başka birşeye göre değişimi girdiği zaman aslında türevden bahsediyoruzdur, değil mi?
- Her nöronun girdisi aslında başka bir nöronun çıktısının ağırlıklarla çarpılmış halidir.
- Dolayısıyla her nöronun çıktısı, o nörona gelen ağırlıkların bir fonksiyonudur. Dolayısıyla hata da ağırlıkların bir fonksiyonu olarak düşünülebilir.

Backpropagation

- Ağırlığı hangi yönde değiştireceğimiz görmek için hatanın bulunduğu nörona gelen bütün ağırlıklara göre değişimini bulmak isteriz. Bu da şu şekilde bulunur:

$$\frac{\partial E}{\partial w_{ij}}$$

- Bu *partial derivative'*in sonucu bize w_{ij} ağırlığına göre hatanın ne kadar ve ne yönde değiştiğini söyler.

Backpropagation

- Bu çok önemli bir ipucudur. Çünkü bu değeri ağırlığa ekleyerek ya da çıkararak hatanın istediğimiz yönde artmasını ya da azalmasını sağlayabiliriz.
- Peki ama hata yani *E* dediğimiz şey de ne?
- Hatayı hesaplamanın birçok yolu var.
- Bu noktada örneğin kolay olmasını sağlamak için şöyle bir hata fonksiyonu tanımlayalım:

$$E = \sum_{i} (y_i - \hat{y}_i)^2$$

Backpropagation

- O zaman az önceki *partial derivative'*i tekrar yazalım.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{n} \left(y_n - \hat{y}_n \right)^2$$

- Burada önemli bir sadeleştirme yapmamız gerekiyor. O da $\hat{y}_n$ sadece kendisine bağlı ağırlıklardan etkileneceği için bağlantısı olmayan bütün nöronları denklemden çıkarabiliriz. Bu da toplam işleminin denklemden tamamen çıkarılabileceği anlamına gelmektedir.

Backpropagation

- Ağırlık güncelleştirme denklemimizin son hali şu şekilde olmuştur.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} (y_j - \hat{y}_j)^2$$

- Burada *y* ve *ŷ* değişkenlerinin indislerinin nasıl değiştiğine bakın. Şimdi bu türevi hesaplayalım.

$$\frac{\partial E}{\partial w_{ij}} = -2(y_j - \hat{y}_j) \cdot \frac{\partial \hat{y}_j}{\partial w_{ij}}$$

Backpropagation

- Artık herşey çok daha net. Çünkü $\hat{y}_j$'in bir nöronun çıktısı olduğunu biliyoruz.
- Hatırlarsanız bir nöronun çıktısı ona gelen bütün inputların toplamının bir aktivasyon fonksiyonuna verildikten sonraki haliydi. O zaman biz de bu terimleri yerlerine koyalım. Örnek teşkil etmesi açısından da aktivasyon fonksiyonu olarak türevi kolay alınan bir fonksiyon olan *sigmoid*'i seçelim.

$$\frac{\partial E}{\partial w_{ij}} = -2(y_j - \hat{y}_j) \cdot \frac{\partial}{\partial w_{ij}} \sigma\left(\sum_i w_{ij} \cdot \hat{y}_i\right)$$

Backpropagation

- Bu noktada artık daha ilerlemeye gerek yok.
- Çünkü bir fonksiyonun türevini cebirsel olarak alabileceğimiz gibi tamamen nümerik bir şekilde de hesaplayabiliriz.
- Şimdi biraz bu kaotik gözüken denklemin sonucunda ne bulduğumuzdan bahsedelim.
- Bulduğumuz şey en basit söylemle NN’leri eğitmenin anahtarı.

Backpropagation

- Bize hangi yönde ilerlersek hatanın azalacağını ya da artacağını söyleyen ve yaptığımız hatanın boyutu hakkında bilgi sağlayan bir *“magic expression”.*
- Ama eğer burada söylenen miktarda hareket edersek hedeflediğimiz yerin üzerinden atlayıp kaçırmamız söz konusu olabilir. Bundan dolayı bu miktarın yalnızca bir kısmını alırız. Bu durumu şu şekilde ifade edebiliriz.

$$w_{ij} = w_{ij} - \eta \cdot \frac{\partial E}{\partial w_{ij}}$$

Backpropagation

- Burada *η*'ya *learning rate* denir.

$$w_{ij} = w_{ij} - \eta \cdot \frac{\partial E}{\partial w_{ij}}$$

- Bize hesaplanan hatanın ne kadarını alacağımızı söyler.
- *η*'nın değeri probleme göre değişiklik gösterir.
- Ancak hemen her zaman 1'in altındadır.

Vanilla Deep Neural Networks

- Az önce konuştuğumuz konular ile bir DNN'nin neye benzediğini ve hem DNN'lerin (hem de sığ ANN'lerin) nasıl eğitildiklerini biliyoruz.
- O zaman artık daha ilginç ağların yapılarını bakmanın zamanı geldi.
- Buna *Convolutional Neural Network*'lere bakarak başlayacağız.

Convolutional Neural Networks

- *Convolutional Neural Networks (CNN)* yapay sinir ağlarının sinyallerle işlem yapması için özelleştirilmiş bir alt grubudur.
- Bu sinyaller:
  - 1D (i.e. ses)
  - 2D (i.e. resim)
  - 3D (i.e. video)

  olabillir.

Convolutional Neural Networks

- Ama CNN'lerin esas parlaması 2012 yılında ImageNet yarışmasında AlexNet adı verilen bir ağın en iyi sonuçları vermesiyle olmuştur.
- AlexNet'i geliştirenler arasında belki daha önceden ismini hatırlayacağınız Geoffrey Hinton bulunmaktadır.
- Merak edenler makaleyi burada bulabilir:
  - ImageNet Classification with Deep Convolutional Neural Networks

Convolutional Neural Networks

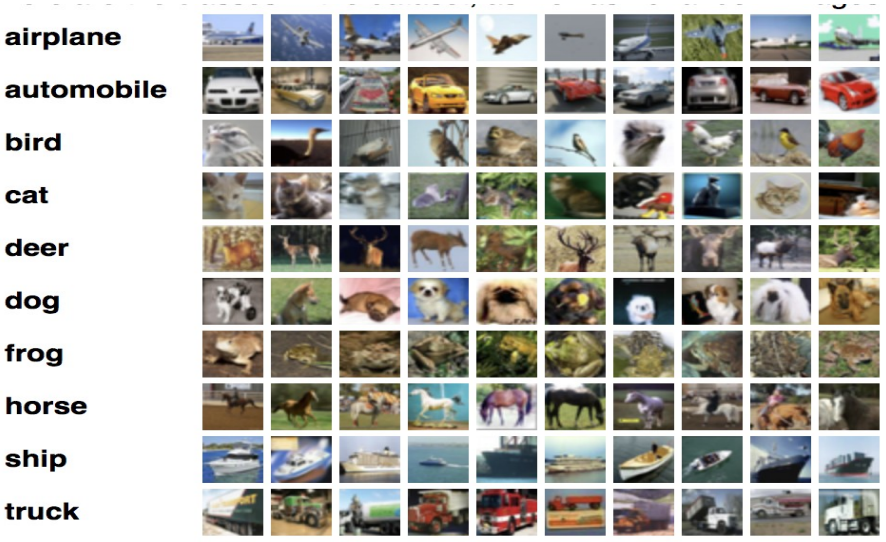
- Şimdi biraz CNN'lerin neden bu kadar iyi sonuç verdiklerinden bahsedelim.
- Normal bir *computer vision* görevi şu alt görevlerden oluşur:
  - Sınıflandırma (i.e. dog / cat)
  - Obje bulma (i.e. resimde bulunan objeleri tanımlayarak dikdörtgen içine alma)
  - Resim bölümleme[*image segmentation*]

  ve daha bir çoğu.

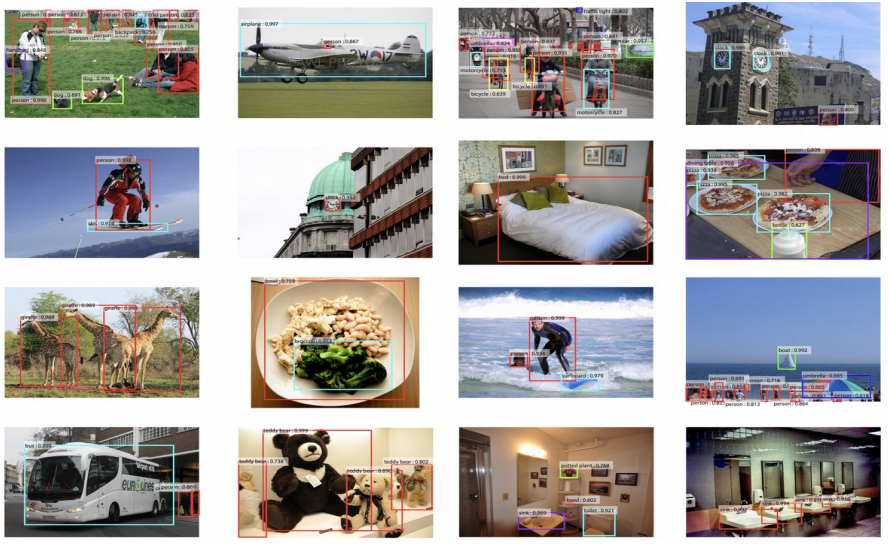
Convolutional Neural Networks

- Bilgisayar bir resmi üst ve alt sınırı belli sayılardan oluşan bir matrix şeklinde görür.
- Bu matrixin belli yerlerinden bazı işlemler yaparak sayısal değerler türetmemiz gerekir.
- Aşağıdaki linkte CNN'lerin kullanıldığı çok sayıda örnek ve açıklama bulabilirsiniz:
  - 9 Applications of Deep Learning for Computer Vision

Convolutional Neural Networks



Sınıflandırma



Obje Bulma

Convolutional Neural Networks

- En basit düzeyde CNN bir resimden belli başlı özellikleri çıkarır.
- Bunu yapmanın en bilinen yolu filtre kullanmaktır.
- Bir filtre resmin üzerinde gezdirilerek yeni bir resim (özellik matrixi) üretmek üzere tasarlanmış daha küçük bir matrixtir.
- Filtrelerin boyutları 3x3, 5x5, hatta bazı durumlarda resimle aynı boyutta bile olabilir. Bu boyuta *kernel* denir.

Convolutional Neural Networks

- Filtrenin resme uygulanma işlemine *convolution* denir.
- (Formalite olarak) *Continuous 1D Convolution* işlemi şu şekilde tanımlanmıştır:

$$(f*g)(t)=\int_{-\infty}^{\infty} f(\tau)\,g(t-\tau)\,d\tau$$

- Bu denklemde *f* de *g* de birer fonksiyondur. *Convolution* işleminde *f* input sinyalini *g* de *kernel* fonksiyonunu temsil eder.
- Kernel fonksiyonundan kastımız aslında filtrenin kendisidir.
- Şimdi daha matematiksel olarak derine girmeden, görsel olarak aslında neden bahsettiğimize bir göz atalım.

Convolutional Neural Networks

- Bir boyutlu bir sinyali ele alalım (*f*):

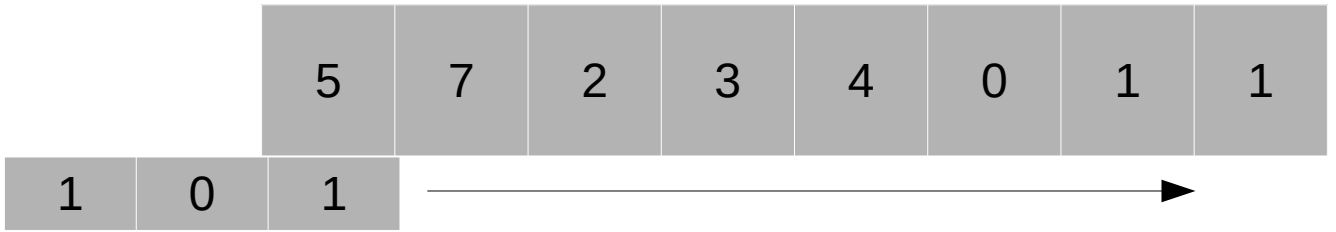
5	7	2	3	4	0	1	1

- Bir tane de bir boyutlu filtre oluşturalım *(g):*

1	0	1

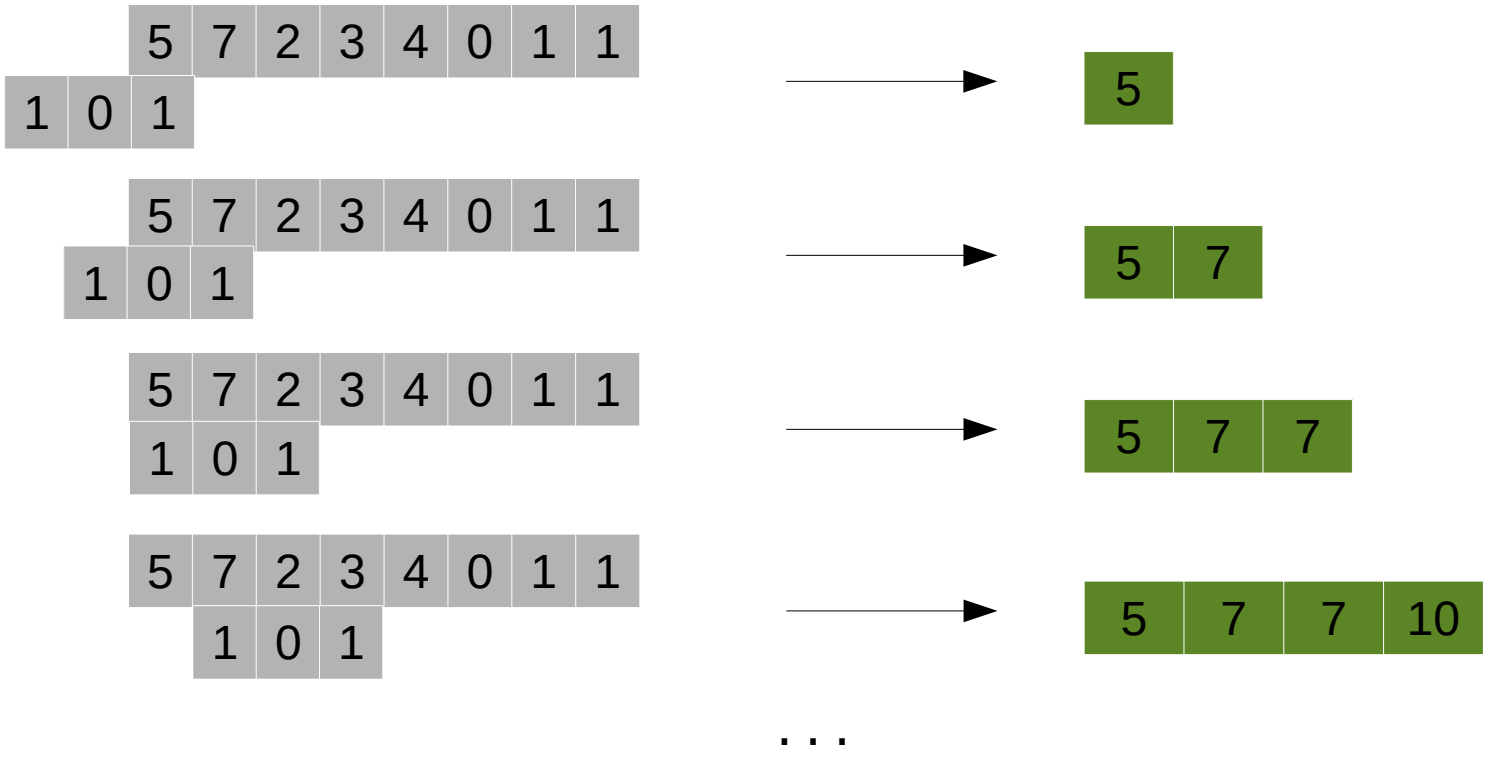
Convolutional Neural Networks

- O zaman *f* * *g* işlemini şu şekilde görselleştirebiliriz.



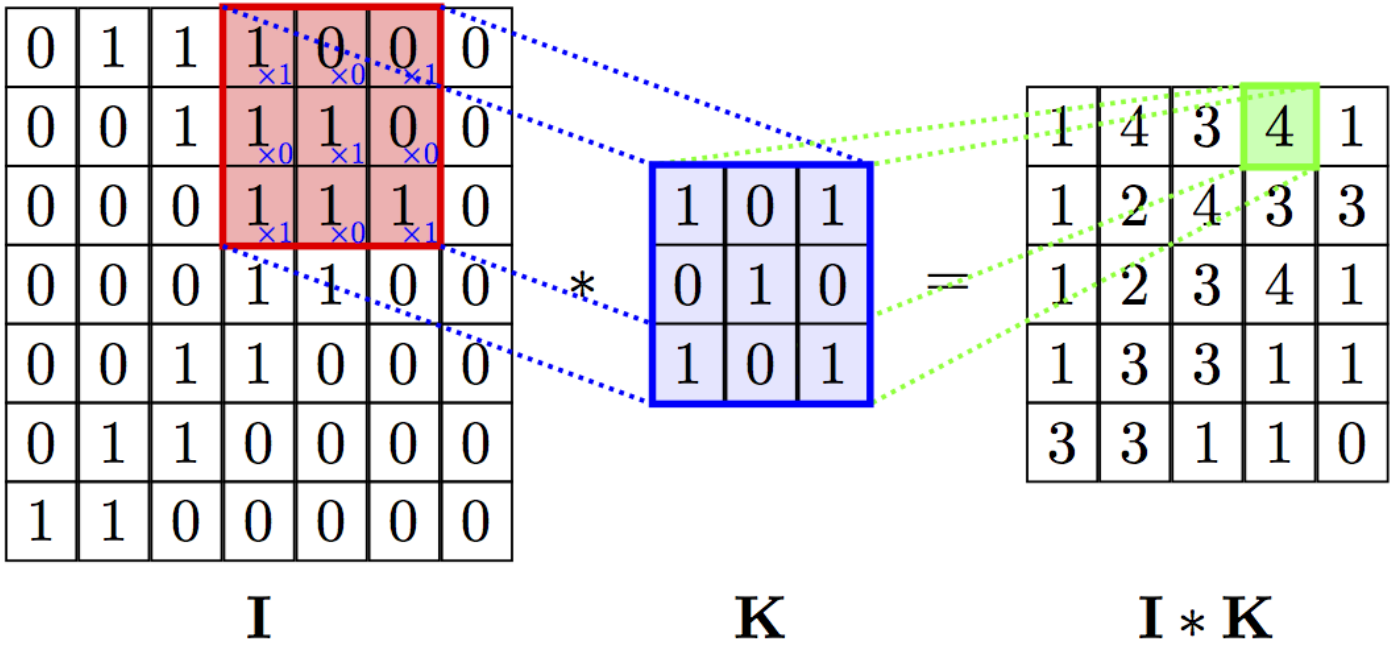
- Üst üste gelen her hücreyi bir biriyle çarpıp toplayarak yeni bir sinyal üretiriz. Yukarıdaki şekli devam ettirirsek:

Convolutional Neural Networks



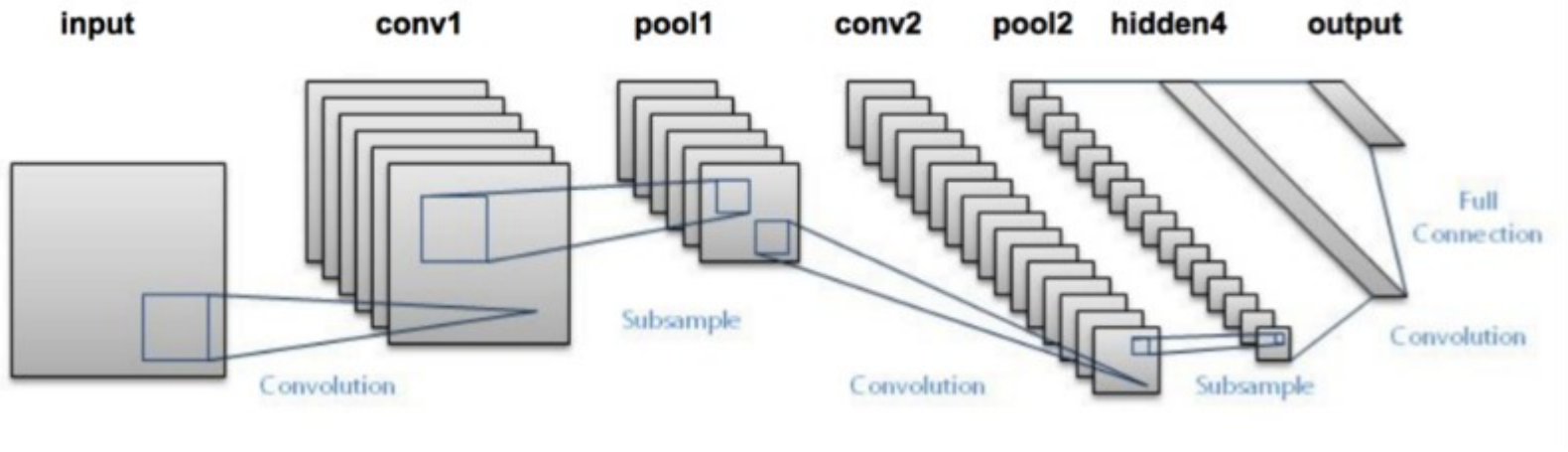
Convolutional Neural Networks

- Tam olarak aynı işlemi iki boyutlu sinyallere de uygulayabiliriz. Örneğin:

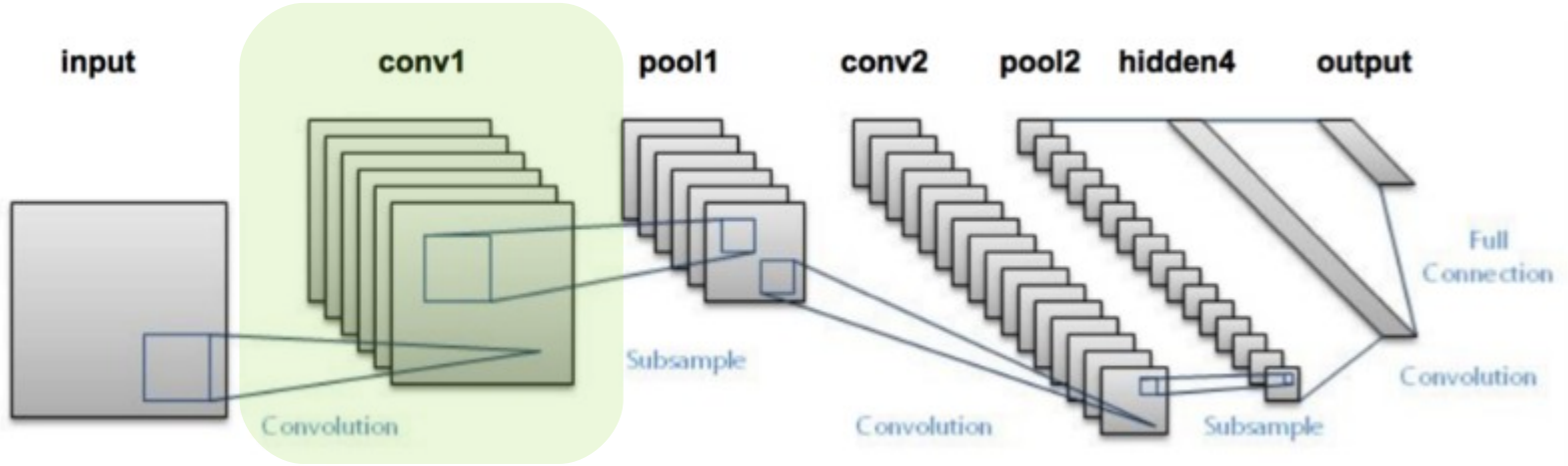


Convolutional Neural Networks

- Şimdi *convolution* işlemini anladığımıza göre, artık CNN'lerin bu filtreleri ve bu işlemi kullanarak ne yaptığına bir göz atabiliriz. Buna bir CNN mimarisine göz atarak başlayalım.

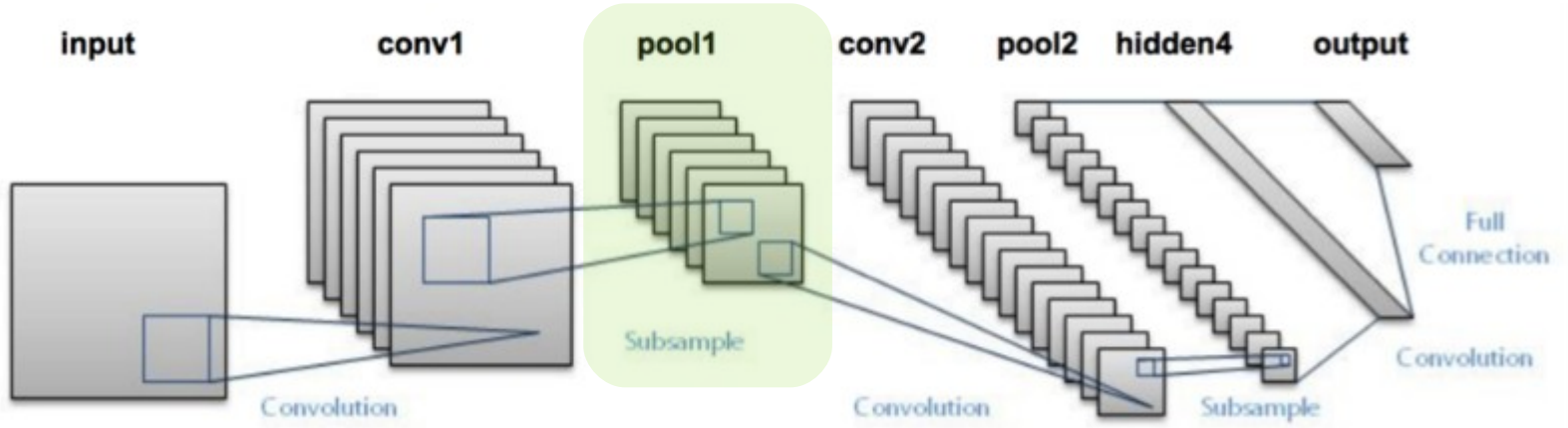


Convolutional Neural Networks



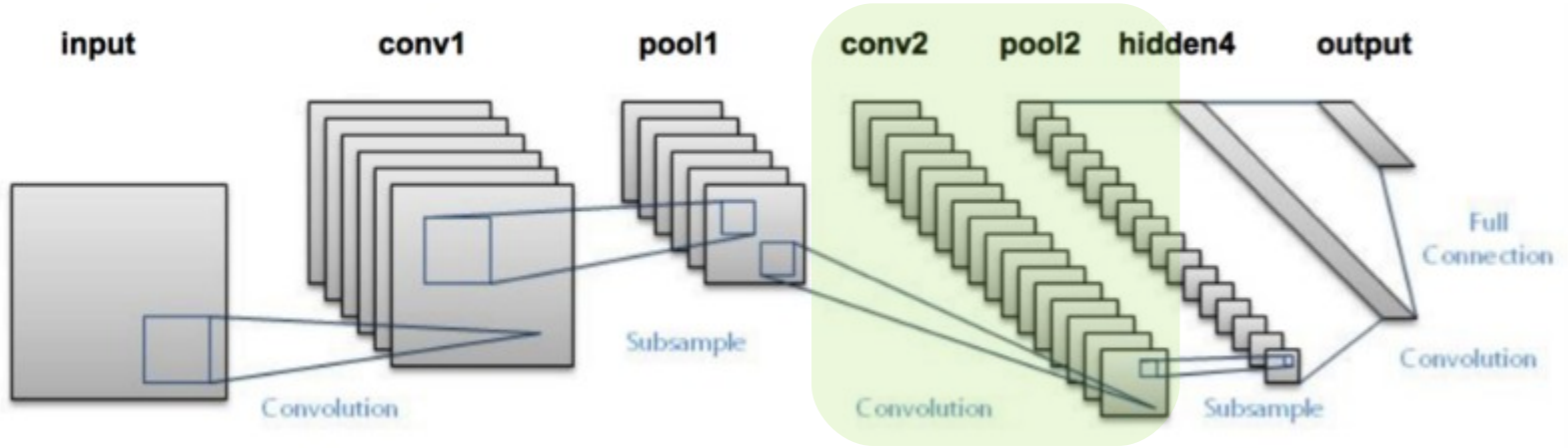
- Conv1 katmanında gördüğünüz her bir kare bir filtreyi temsil etmektedir.
- Hatırlayacağınız gibi bir filtreyle bir resmin *convolve* edilmiş hali yeni bir resim olduğundan bu katmanın sonunda filtre sayısı kadar yeni filtrelenmiş resim elde edilecektir.

Convolutional Neural Networks



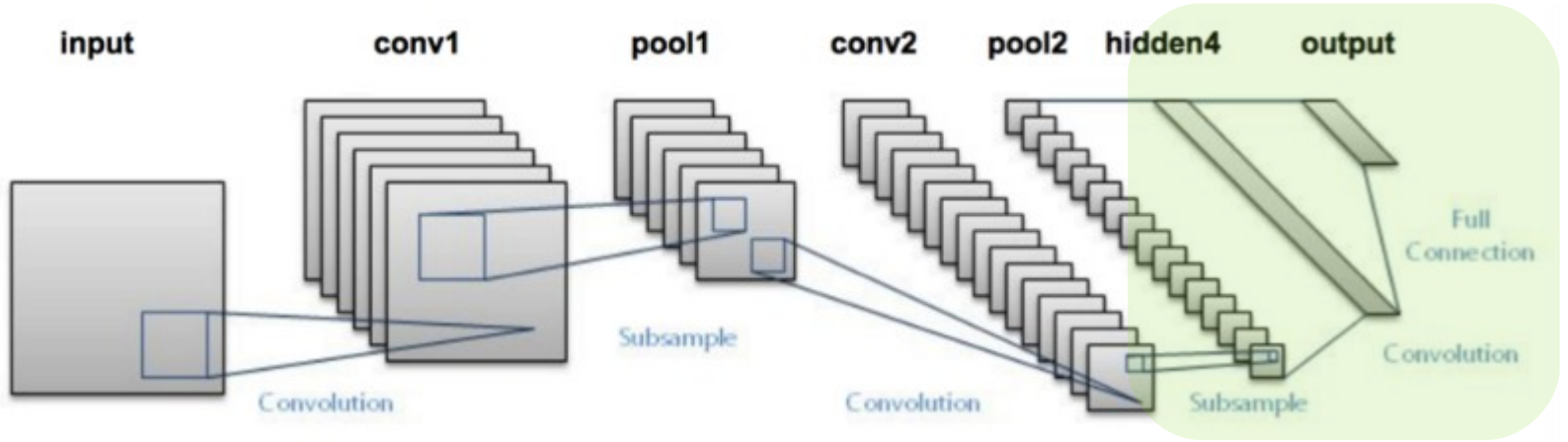
- Daha sonra gelen pool1 katmanın bir önceki katmandan gelen resimlerin çözünürlüklerini düşürür. Bu katmanın teknik adı Pooling'dir. (MaxPooling)

Convolutional Neural Networks



- Sonraki yeniden bir adet convolutional katman ve bir adet MaxPool katmanı yer almıştır.

Convolutional Neural Networks



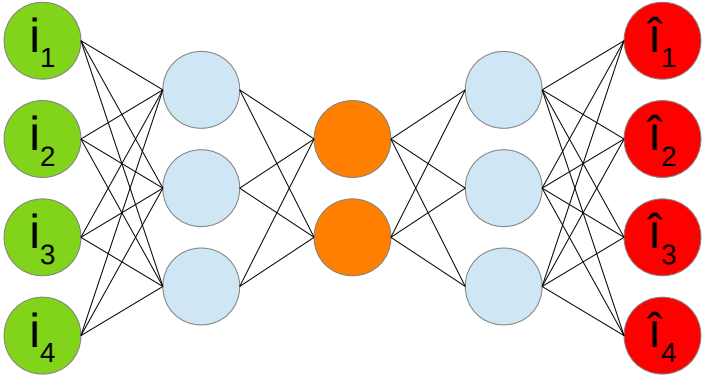
- En son katman(lar) da bildiğimiz standart derin ya da sığ bir neural networktür. [*Fully Connected / Dense*]
- Bu kısmın amacı CNN tarafından çıkarılmış özellikleri kullanarak bir sınıflandırma (ya da başka birşey) yapmaktır.

Convolutional Neural Networks

- Yine meşhur sorumuza geri döndük: Peki öğrenme bu mimarinin nerelerinde yapılıyor?
- Burada sıradan bir neural network'e göre farklılaşma filtrelerde oluyor.
- İlk olarak, aynı bir ağırlığın küçük hareketlerle değiştirilmesi gibi filtreler de doğru sonuca ulaşmak için küçük hareketlerle değiştiriliyor.
- İkinci olarak da son katmanlardaki *fully connected* bölümün ağırlıklarında yapılıyor.

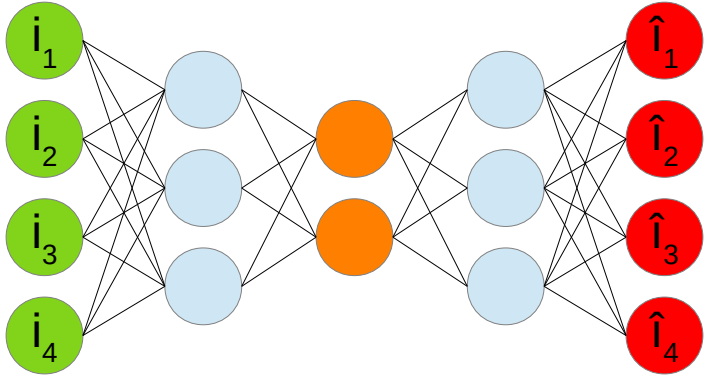
Autoencoder Networks

- Şimdi biraz da Autoencoder'lara bakalım.
- İlk bakışta bir Autoencoder'ın görünüş olarak standart bir derin neural network'ten hiç bir farkı yoktur. Aşağıda görebilirsiniz.



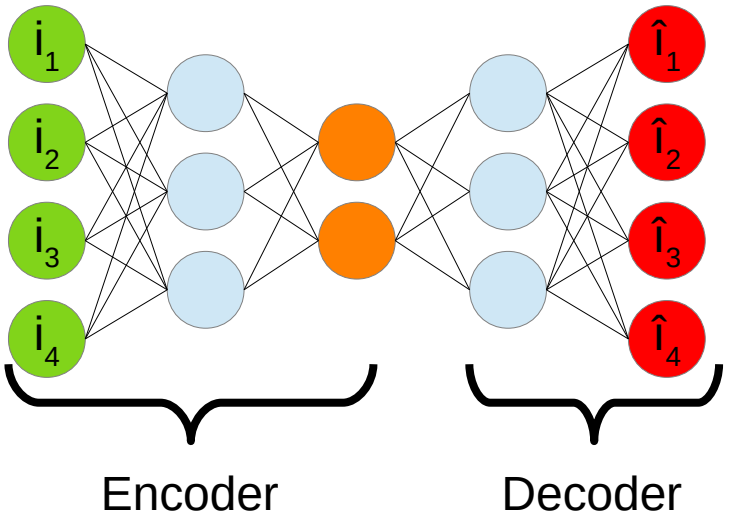
Autoencoder Networks

- Bir autoencoder aldığı girdiyi aynı şekilde çıkarmak için eğitilen bir DNN'dir.
- Ancak ortadaki (turuncuyla gösterilen) daha küçük alana dikkat edin.



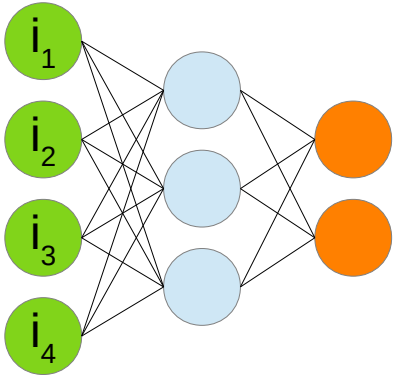
Autoencoder Networks

- Başlangıçta 4 boyut olarak verilen veri, o noktada 2 boyuta kadar sıkıştırılır.
- Dolayısıyla bir autoencoder'a iki ayrı parçaymış gibi bakabilirsiniz:
  1) Encoder (4 boyuttan 2 boyuta indiren kısım)
  2) Decoder (2 boyuttan tekrar 4 boyuta çıkaran kısım)



Autoencoder Networks

- Daha sonra Encoder kısmı ağın geri kalan kısmından ayrılır ve tek başına kullanılmaya devam edilir.
- Bu kısım yanda da göreceğiniz gibi 4 boyutlu olarak eğitildiği veriyi 2 boyutlu olarak ifade edebilir.



Autoencoder Networks

- Ağın bu kısmının arkasına artık istediğiniz herşeyi bağlayabiliriniz.
- Bu yeni bir *fully connected* ağ da olabilir, tamamen farklı bir makine öğrenmesi yöntemi de olabilir.
- Autoencoder'ların kullanıldığı bazı alanlar şunlardır:
  - Sinyal Parazitsizleştirme [*Signal De-noising*]
  - Boyut İndirgeme [*Dimensionality Reduction*]
  - Sinyal Üretme [*Signal Generation*]