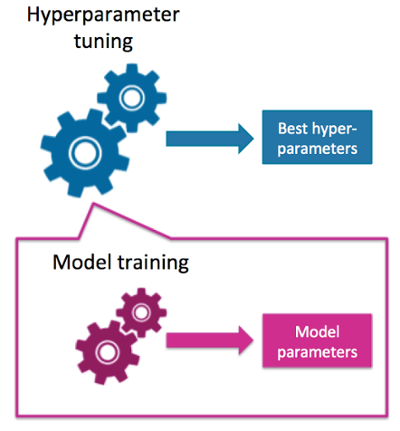
**Derin öğrenmeyle birlikte makine öğrenmesindeki çalışmalar öznitelik mühendisliğinden mimari mühendisliğine doğru evrildi.** Daha önceki çalışmalarda araştırmacılar çoğunlukla ilgili problemi çözebilmek için problemi temsil edecek en iyi öznitelik kümelerinin oluşturulması; yani özniteliklerin çıkarılması ve bu öznitelikler arasında temsil kabiliyeti en yüksek olanların seçilmesi üzerinde çalışıyorlardı. Derin öğrenme yaklaşımlarının gelişimi ile birlikte çok katmanlı yapay sinir ağının nasıl tasarlanacağı; kaç katmandan oluşacağı, ne kadar nöron içereceği, hangi optimizasyon algoritmasının ya da aktivasyon fonksiyonunun kullanılacağı problem çözümünde daha önemli hale geldi.

Grafik kartları ile paralel işlemlerin daha hızlı yapılabilmesi araştırmacıların daha fazla katmandan oluşan daha karmaşık yapılar tasarlayabilmesine imkan vermesiyle birlikte; **derin öğrenme ile problem çözmek çok katmanlı ağ yapısını en iyi ve optimum şekilde tasarlamak ile eşdeğer duruma geldi.** İşte bu mimari mühendisliğinde araştırmacının sezgisinden sonra en sık kullandığı araçlar hiper parametreler oldu.

**Hiper-parametre Nedir?**

Veriden öğrenen makine öğrenmesi modelleri tasarlanırken, modelde kullanılan algoritma ya da teknikler ne olması gerektiğine tasarımcının karar vermesi gereken bazı parametreleri de beraberinde getirmektedir. Örneğin, KNN sınıflandırma algoritmasında k değerinin ne olacağına modeli tasarlayan kişi karar vermektedir. Benzer şekilde SVM algoritmasında hangi kernel fonksiyonunun kullanılacağına da tasarımcı karar verir. Aynı şekilde derin ağ modellerinde seyreltme (dropout) değerine, katman sayısına, nöron sayısına modeli tasarlayan kişi karar vermektedir. Genelde bu parametreler için hangi tercihte bulunulacağı başlangıçta açık ve kesin değildir; probleme, verisetine vb. gibi etmenlere göre değişiklik göstermektedir. Bu nedenle parametrelerin seçimi modeli tasarlayan kişiye bırakılmıştır. **Ne olması gerektiği, modeli tasarlayan kişiye bırakılmış, probleme, veri setine göre değişiklik gösteren parametreler hiper-parametre (hyperparameters) olarak adlandırılmaktadır.**



Şekil 1 : Derin Öğrenme Uyglamalarında Parametre ve Hiper parametre (Resim Kaynak : <http://www.xlgps.com/article/317732.html>)

Modelin yüksek başarım sağladığı birbirinden farklı hiper parametre grupları olabilmektedir. Bu grupların her birinin model tasarımında kullanılmasında sakınca yoktur. Bununla birlikte en uygun hiper parametre grubunun seçilmesi baş edilmesi gereken önemli problemlerden biridir. Hiper parametrelerin seçimi genelde tasarımcının sezgisine, daha önceki problemlerden elde edilen tecrübelere, farklı alanlardaki uygulamaların kendi problemimize yansımasına, güncel trendlere, model içindeki tasarım bağımlılığı vb. göre değişmektedir. Bununla birlikte son zamanlarda problemin çözümü için en uygun hiper parametre grubunun en optimum şekilde seçilmesine yönelik farklı teknikler ortaya atılmaktadır.

Bu yazı serisinin ilk bölümünde öncelikle derin öğrenme uygulamalarında en sık kullanılan hiper parametreler değerlendirilecek olup daha sonra hiper parametre seçim yöntemleri üzerinde durulacaktır. Son olarak da derin öğrenme temelli çalışmaların hiper parametrelerinin analiz bölümü üzerine küçük bir öneri kısmı olacaktır.

**Derin Öğrenme Uygulamalarında En Sık Kullanılan Hiper Parametreler**

**Veri Setinin Boyutu**

Derin öğrenme uygulamalarında veri setinin büyüklüğü ve çeşitliliği öğrenme için en önemli faktör olduğu artık hepimizin malumu. Veri setimiz ne kadar büyük olursa öğrenme o oranda iyi olacaktır. Bununla birlikte veri seti büyüklüğü öğrenme için harcanan zamanı ve öğrenme sonunda elde edilen modelin büyüklüğünü de o oranda arttırmaktadır. Çok sık eğitilmeyecek ve depolama alanı sorun etmeyecek projelerde öğrenim başarısına karşı zaman ve depolama alanındaki bu kayıplar gözardı edilebilmektedir. Bununla birlikte sık sık eğitilmesi gereken veya mobil ortamlar gibi depolama alanı problem oluşturacak ortamlarda çalışacak projeler geliştiriliyorsa başarımın yanı sıra veri setinin büyüklüğü de değerlendirmeye alınmalıdır.

Sadece veri setinin büyüklüğü iyi bir model için yeterli değildir, bunun yanısıra çeşitliliği de önemlidir. Çeşitlilik arttığı oranda modelin başarımı da artış gösterecektir.

Veri seti arttıkça başarım sonsuza kadar doğru orantılı olarak artmaz, belli bir noktadan sonra başarım küçük küçük artmaya başlar; veri setinin temsiliyetinin de bir sınır noktası vardır. Bu anlamda modeli eğitirken elimizdeki veri setinin hangi noktada kırıldığı da araştırılmalıdır. Yani aynı başarımı daha küçük veri seti kullanarak elde edebiliyorsak ve eğitim, depolama alanı bizim için önemliyse veri setinden belli oranda feragat edilebilir.

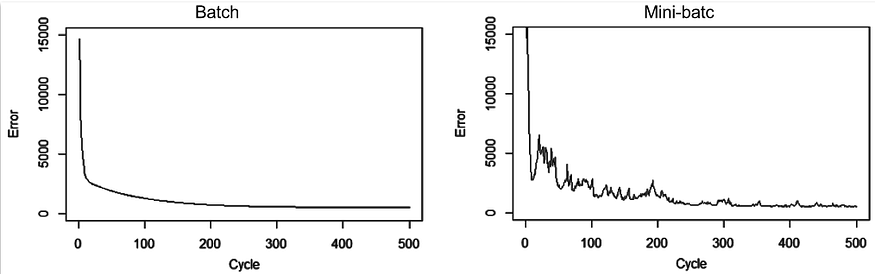
Veri seti ile ilgili bazı özellikler aşağıdaki gibidir;

* Elimizde çok küçük veri seti varsa problem derin öğrenme ile çözümü çok uygun değildir. Sentetik veri ile veri seti artırılmalıdır ya da “transfer learning” yöntemleri ile öznitelik transferi yapılmalıdır. Ya da küçük veriler üzerinde daha iyi çalışan doğrusal olmayan (“non-linear”) problemlerin çözümü için ideal olan SVM gibi modeller kullanılmalıdır.
* Görsel imajlar üzerinde çalışıyorsak onlarca, yüzlerce veri modelin başarımı için çok da yeterli değildir. Her bir sınıf için elimizde en az binlerce veri olması gerekiyor.
* Veri setindeki sınıflar arasında geçişkenlik fazlaysa; yani sınıflar bir birine çok fazla benziyorsa ya da çok fazla gürültülü veri varsa, başarım grafiği çok fazla inişli çıkışlı bir grafik sergileyebilir. Bu durumda eğer veriyi çok fazla değiştirme şansımız yoksa “batch” değerini değiştirerek problemi çözebiliriz.
* Veri seti sentetik veri üretme teknikleriyle artırılabilir. Başarımı belli oranda artıracaktır.

**“Mini-Batch” Boyutu**

Derin öğrenme uygulamalarında, veri setinde bulunan tüm verileri aynı anda işleyerek öğrenme, zaman ve bellek açısından maliyetli bir iştir. Çünkü öğrenmenin her iterasyonunda geriyeyayılım (“backpropagation”) işlemi ile ağ üzerinde geriye dönük olarak gradyan (“gradient descent”) hesaplaması yapılmakta ve ağırlık değerleri bu şekilde güncellenmektedir. Bu hesaplama işleminde veri sayısı ne kadar fazla ise hesaplama da o oranda fazla sürmektedir. Bu problemi çözmek için; veri seti küçük gruplara ayrılmakta ve öğrenme işlemi seçilen bu küçük gruplar üzerinde yapılmaktadır. **Bu şekilde birden fazla girdinin parçalar halinde işlenmesi “mini-batch” olarak adlandırılmaktadır. Model tasarlanırken mini-batch parametresi olarak belirlenen değer; modelin aynı anda kaç veriyi işleyeceği anlamına gelmektedir.**

**Yapılan çalışmalarda bu şekilde verilerin gruplar halinde işlenmesi (mini-batch) durumunda loss değerinin arttığı fakat zamandan kazanıldığı tespit edilmiştir.**



Şekil 2 : Batch ve mini-bath uygulandığı durumda hata değeri değişimi (Resim kaynak : [Steven Schmatz](https://www.quora.com/In-a-neural-network-unsure-if-it-is-keras-specific-what-is-the-difference-between-batch-size-and-batch-training-When-would-I-need-both))

Mini-batch uyguladığımızda Şekil 2'deki grafikte de görüleceği üzere hata değerinde dalgalanmalar olur. Çünkü her iterasyonda farklı veri işlendiği için bazı iterasyondaki veriler için seçilen parametreler tam uygun iken bazısında uygun olmayabilir. Bununla birlikte iterasyon arttıkça bu zikzaklar azalacaktır. Mini-batch durumunda ortaya çıkan bu gürültü durumunu azaltmanın bir yolu da küçük öğrenem değeri (learning rate) belirlemektir.

Mini-batch değerini [1, eğitim\_kümesindeki\_tüm\_verilerin\_sayısı] kapalı aralığında bir değer olarak belirleyebiliriz. Mini-batch değerini alabileceği en küçük değer olan 1 olarak belirlediğimizde “schoastic gredient decent” ile aynı işi yapmış oluruz. Yani her iterasyonda sadece tek bir veri üzerinde işlem yaparız. Bu durumda;

* Modelin gürültüyü öğrenmesine neden olabilir. Yani grafikteki zikzaklar çoğalır. Çünkü öğrenim sürecinde her defasında farklı veri kullanılmaktadır.
* Local optimum’da takılıp global optimuma hiç ulaşamayabilir.
* Optimizasyon algoritması yanlış yönlere gidebilir. Bazen doğru yollardan gidebileceği gibi bazen yanlış yollardan giderek zaman kaybına neden olabilir.
* Hiç bir zaman global minumuma ulaşmadan, onun etrafında dolanabilir.
* Özetle “schoastic gradient descent” optimizasyon işlemi uzun zaman alan bir süreç olacaktır.
* **Momentum ile “schoastic gradient descent”in salınımları azaltılarak daha hızlı ve tutarlı bir optimizasyon algoritması oluşturulabilir.**
* Salınımı azaltmanın bir diğer yoluda RMSprob (root mean square error propability) kullanmaktır.

Mini-bath değerinin eğitim kümesinde tüm elemanların sayısına eşit olduğu durumda; eğitim kümesindeki tüm veriler eğitime gireceği için yapılan işlem “batch gredient descent”le aynı olacaktır. Yani Şekil 2'de soldaki grafiğe benzeyecektir. Böyle bir durumda;

* Model daha az gürültü öğrenecektir.
* Bununla birlikte aynı anda bütün veriyi işleyeceği için öğrenme çok uzun sürecektir.
* Optimizasyon işleminde büyük adımlarla ilerleme olacaktır.

**Mini-batch değerinin seçiminde en uygun değer 1 ile eğitim kümesindeki tüm verilerin sayısı arasında ne çok küçük ne de çok büyük olmayan bir değer belirlenmelidir.** Bu hızlı şekilde öğrenmeyi sağlayacaktır. Mini-batch boyutunun bu şekilde belirlenebilmesiyle ile ilgili bazı özellikler aşağıdaki gibidir;

* Batch size’ın büyük olması, daha doğru gradyan değerinin hesaplanmasını sağlamaktadır. Bu durum da linerizasyonu azaltmaktadır.
* Belirlenen batch değerinin GPU belleğine sığması gerekiyor. Bu nedenle batch boyutu 2’nin katları şeklinde belirlenmelidir; 2, 4, 8, 16, 32, … 512 vb. Bu şekilde belirlenmemişse başarımda ani düşüşler yaşanabilir.
* Batch size genelde 64ile 512 arasında 2'nin katı olan değerlerden belirleniyor.
* Eğitim kümemdeki elaman sayısı küçükse (yani 2000'den az ise) eğitim kümesindeki tüm elemanlar aynı anda kullanılabilir. Yani batch gradyan hesaplaması yapılabilir.
* Evrişimsel sinir ağları (Convolutional Neural Networks) batch değerine karşı hassastır. Batch değerindeki küçük değişiklikler başarımda büyük etkiler oluşturabilir.
* Batch boyutunun diğer bir kıstası da bellek boyutudur. Eğer küçük belleğe sahip ortamda çalışıyorsanız, batch büyük tutmakta zorlanabilirsiniz. Bu nedenle modeli tasarlarken öncesinde kullanabileceğiniz maksimum batch değeri hesaplamak verimli olacaktır.
* Hessian’da batch size’lar büyük olmalıdır; örneğin 10.000 gibi.
* Batch size küçük olması iyileştirme (reguralization) etkisi yaratmaktadır. Modele veri büyük gruplar halinde verildiğinde ezberleme daha fazla oluyor.
* Batch işleminde, veri seti batch değeri olarak belirlenen değere göre parçalara ayrılmakta ve her iterasyonda modelin eğitimi bu parça üzerinden yapılmaktadır. Bununla birlikte bazı durumlarda veri kendi içinde gruplanmış olabilmektedir. Bu durum veri seti içinde korelasyon oluşturacak; bu veri setinden seçilecek test setin de yüksek başarım vermesini sağlayacak böylece ezberleme (“overfitting”) olacaktır. Bunu önlemek için eğitim başlamadan veri seti parçalara ayrılmadan önce veriseti karıştırılmalıdır (shuffle). **Batch seçiminde verilerin rastgele seçilmesi önemlidir.**

**Öğrenme Hızı (Learning Rate) ve Momentum Katsayısı (Momentum)**

Derin öğrenmede parametrelerin güncellenmesi geriyeyayılım (“backpropagation”) işlemi ile yapılmaktadır. Backpropagation işleminde bu güncelleme işi “chain rule” olarak adlandırılan geriye doğru türev alarak farkın bulunması ve bulunan fark değerinin “learning rate” parametresiyle çarpılması, çıkan sonucun ağırlık değerlerinden çıkarılarak yeni ağırlık değerinin hesaplanmasıyla yapılmaktadır. Bu işlem esnasında kullanılan “learning rate” parametresi sabit değer olarak belirlenebilir, ya da adım adım artan bir değer olarak da belirlenebilir (örneğin belli bir öğrenme adımına kadar 0.001 o adımdan sonra 0.01 gibi), momentum değerine bağlı olarak belirlenebilir ya da adaptif algoritmalar tarafından öğrenme esnasında öğrenilebilir.

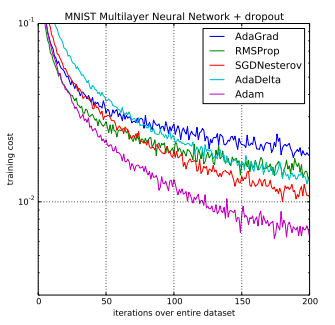
Schoastic gradient descent gibi çok fazla salınım oluşturan, gürültü üreten yöntemler üssel ağırlık ortalaması (exponential weight avarage) gibi tekniklerle normalize edilip salınımları azaltılmaktadır. Bu normalizasyon işlemi momentum beta katsayıları ile yapılmaktadır. Bu teknikte yeni üretilen değeri olduğu gibi almak yerine, bir önceki değeride beta katsayısı oranında işin içine katarak yeni değer hesaplanır. Böylece gürültü ve grafikteki salınımlar azaltılarak daha hızlı bir yöntem oluşturulur.

Öğrenme hızı ve momentum ile ilgili bazı özellikler aşağıdaki gibidir;

* Öğrenme hızı yüksek tutuyor olmam veriden çok etkileniyorum demektir.
* Öğrenme hızı yüksek olması salınıma neden olacaktır. Buna karşı küçük olması da küçük adımlarla ilerleyeceğinden öğrenimin çok uzun sürmesine neden olacaktır.
* Öğrenme hızı için en uygun çözüm başlangıçta öğrenme hızı yüksek tutmak, gittikçe azaltmaktır.
* Öğrenme hızı başlangıçta çok küçük olması, lokal optimum değere takılarak, global optimum değere hiç ulaşılamamasına neden olabilir.
* Öğrenme hızı değeri genelde varsayılan değer olarak 0.01 kullanılmakta belli bir epoch’dan sonra 0.001'e düşürülmektedir.
* Schoastic gredient descent momentum ile kullanıldığında hızı artırmakta, salınımları azaltmaktadır.
* Genelde kullanılan momentum beta katsayısı 0.9'dur. Uygun parametre aralığı ise 0.8 to 0.99'dur.

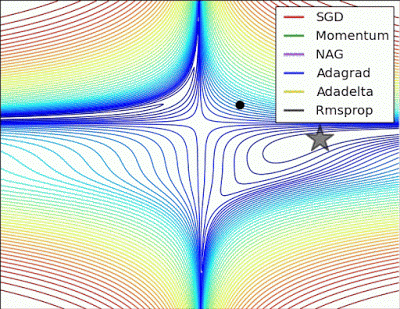
**Optimizasyon Algoritmasının Seçimi**

Derin öğrenme uygulamalarında öğrenme işleminin temelde bir optimizasyon problemi olduğu daha önce vurgulanmıştı. Doğrusal olmayan problemlerin çözümünde optimum değeri bulmak için optimizasyon yöntemleri kullanılmaktadır. Derin öğrenme uygulamaların yaygın olarak stochastic gradient descent, adagrad, adadelta, adam, adamax gibi optimizasyon algoritmaları kullanılmaktadır. Bu algoriatmalar arasında başarım ve hız bakımından farklılıklar bulunmaktadır.



Şekil 5: Belli başlı optimizasyon algoritmalarının MNIST veriseti üzerindeki çalışma zamanı grafiği

Belli başlı optimizasyon algoritmalarının [MNIST](http://yann.lecun.com/exdb/mnist/) veriseti üzerindeki çalışma zamanı grafiği Şekil 5'deki grafikte görülebilir. Bununla birlikte optimizasyon algoritmalarının optimum değeri bulma yöntemi ve çalışma zamanı ile ilgili daha ayrıntılı bilgi için Sebastian Ruder tarafından 2017 yılında yazılan [“An overview of gradient descent optimization algorithms”](http://ruder.io/optimizing-gradient-descent/) isimli yazıya bakılabilir.



Şekil 6: Optimizasyon algoritmalarının hız performans gösterimi (Resim Kaynak : Alec Radford)

Optimizasyon algoritmalarının bazı özellikleri aşağıdaki gibidir;

* İki tür gradient hesaplama yöntemi var; sayısal (numeric) ve analitik (analytic). Sayısal gradient bize yaklaşık sonuç veriyor, yavaş çalışıyor bununla birlikte uygulaması kolay. Buna karşı analitik gradient hızlı, tam sonuç veriyor ve hataya eğilimli. Pratikte analitik gradient ile hesaplama yapılıyor sonuçlar sayısal gradient ile kontorol ediliyor.
* Derin öğrenme modellerinde genelde varsayalan olarak kullanılan optimizasyon algoritması stochastic gradient descent (SGD).
* Bununla birlikte şekil 6'da görüleceği üzere schoastic gradient descent diğer yöntemlere göre daha yavaş çalışıyor.
* Stochastic Gradient Descent (SGD) bazı problemlerde çok kötü sonuç verebilmektedir; özellikle resim tanıma problemlerinde.
* Adaptive algoritmalar öğrenme hızını (learning rate) kendisi öğrenmektedir ve dinamiktir.
* AdaGrad seyrek parametreler için büyük güncellemeler yaparken sık parametreler için daha küçük güncellemeler yapar. Bu nedenle NLP ve resim tanıma gibi seyrek veriler için daha uygundur.
* AdaGrad’da her parametrenin kendi öğrenme hızı vardır ve algoritmanın özelliklerine bağlı olarak öğrenme oranı giderek azalmaktadır. Bu nedenle öğreneme oranı giderek azalır ve zamanın bir noktasında sistem öğrenmeyi bırakır. Bu AdaGrad’ın en büyük dez avantajıdır.
* RMSprop ve benzeri olan AdaDelta, AdaGrad’ın bu sorununu çözerek bu hızlı düşüşü önler.
* Adam veya adaptif momentum AdaDelta’ya benzer bir algoritmadır. AdaDelta’dan farklı olarak parametrelerin her birinin öğrenme oranlarının yanısıra momentum değişikliklerini de önbellekte (cache) saklar; yani RMSprop ve momentumu birleştirir.
* Adaptif algoritmalar hız açısından SGD’den daha iyi performans göstermektedir. Bununla birlikte SGD momentum ile kullanıldığında, SGD’nin başarım açısından adaptif yöntemlere göre daha yüksek başarım gösterdiği bazı durumlarda var. Örnek için 2017 yılında yayınlanan [“The Marginal Value of Adaptive Gradient Methods in Machine Learning”](https://arxiv.org/pdf/1705.08292.pdf)makalesine bakılabilir.[6]

**Eğitim Tur (Epoch) Sayısı**

Model eğitilirken verilerin tamamı aynı anda eğitime katılmaz. Belli sayıda parçalar halinde eğitimde yer alırlar. İlk parça eğitilir, modelin başarımı test edilir, başarıma göre geriyeyayılım (“backpropagation”) ile ağırlıklar güncellenir. Daha sonra yeni eğitim kümesi ile model tekrar eğitilip ağırlıklar tekrar güncellenir. Bu işlem her bir eğitim adımında tekrarlanarak model için en uygun ağırlık değerleri hesaplanmaya çalışılır. Bu eğitim adımlarının her birine “epoch” denilmektedir.

Derin öğrenmede problemi çözecek en uygun ağırlık değerleri adım adım hesaplandığı için ilk epoch’larda başarım düşük olacak, epoch sayısı arttıkça başarım da artacaktır. Bununla birlikte belli bir adımdan sonra modelimizin öğrenme durumu oldukça azalacaktır.

Modelin eğitilmesi genelde uzun zaman almaktadır; eğitimi günler, aylar süren modeller vardır. Derin öğrenmede bu durum olağandır. Bu nedenle diğer hiper parametreler ile mümkün olduğunca eğitim süreci kısaltılmaya çalışılmaktadır.

Epoch sayısının büyüklüğü problem türüne göre de değişiklik göstermektedir. Örneğin örüntü öğrenilen RNN(Recurrent Neural Network)’lerde epoch sayısı diğer modellere göre daha büyük tutulmalıdır. Epoch sayısı arttıkça modelin başarımı gözle görülür oranda artmaktadır. Başarım belli bir epoch’dan sonra çok küçük birimlerde artış gösterceği için bu noktalarda eğitim sonlandırılabilir.

**Ağırlık (Weight) Başlangıç Değerlerinin Belirlenmesi**

Modelin W ağırlık değerleri model başlatılırken hepsi 0 olacak şekilde, 0.5 gibi standart sapmaya sahip olacak dağılım şeklinde, 0.5 ile 0.9 arasında uniform dağılıma sahip olacak şekilde ya da daha önceki bir modelin ağırlık değerleri kullanılacak şekilde tanımlanabilir. Daha farklı ağırlık belirleme yöntemleri de vardır. Ağırlıkların nasıl belirlendiği modelin öğrenmesine ve hızına etki etmektedir.

Belli başlı ağırlık belirleme yöntemleri ve etkileri aşağıdaki gibidir;

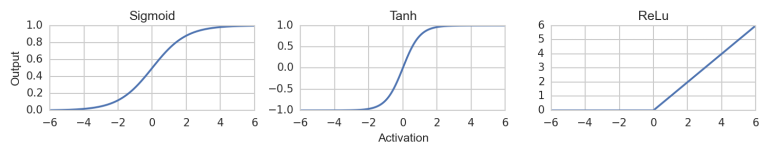
W ağırlıkları modelin başlangıcında 0 olarak başlatılırsa matris çarpımı bir toplam olduğu için, girdi değerleri olduğu gibi çıktı olarak belirir. y = f(x,w) -> w.x olduğundan w matrisinin değerleri 0 olduğu için x’ile sıfır içeren matris toplanması yine x’in kendisini verecektir. Bu nedenle w ağırlıkları başlangıçta 0 olarak başlatılmamalıdır.

W ağırlıkları modelin başlangıcında rastgele küçük sayılardan başlatılırsa model küçük ağlarda çalışır fakat ağ katmanları arasında aktivasyonun heterojen bir şekilde dağılımına neden olur.

**Aktivasyon Fonksiyonu**

Aktivasyon fonksiyonları modele non-linearite katıyor. Gizli katmanlarda (Hidden layer’da) y = f(x,w) şeklindeki lineer fonksiyonumuzda matris çarpımı yapılıp nöronların ağırlığı hesaplandıktan sonra çıktı doğrusal olmayan (non-linear) bir değere dönüştürülür. Çünkü derin öğrenme yöntemleri doğrusal olmayan (non-linear) yapıya sahip problemlerin çözümünde diğer yöntemlere göre daha etkili olduğu için, derin öğrenme yöntemleriyle çözülmeye çalışılan problem genelde doğrusal olmayan non-linear bir problemdir. Matris çarpımı sonucu elde edilen değerin non-linear hale dönüştürülmesi aktivasyon fonksiyonları ile yapılmaktatır. **Aktivasyon fonksiyonları çok katmanlı yapay sinir ağlarında doğrusal olmayan (non-linear) dönüşüm işlemleri için kullanılmaktadır.** Gizli katmanlarda (Hidden layer’larda) geri türev alınabilmesi (gradient decent hesaplanabilmesi) için (öğrenmede fark geri türevle alınıyor) gizli katmanların (hidden layer) çıktısı bazı aktivasyon fonksiyonları ile normalize edilmektedir. Bu aktivasyon fonksiyonlarından bazıları Şekil 3'de fonksiyon grafikleri verilen sigmoid, tanch, ReLu, PreLu vb’dir. **İçlerinde en kullanışlısı ReLu’dur. ReLu’da sigmoid’e göre parametreler daha hızlı bir şekilde öğrenilmektedir.**PReLu ise, ReLU’nun kaçırdığı negatif değerleri yakalamaktadır; **eğer bizim için negatif değerler önemliyse PReLu tercih edilmelidir.**

Aktivasyon fonksiyonlarını gradient descent ile geri dönüp düzeltme yaptığımızda da kullanmaktayız. Bu tarz kullanımdaki amacımız kolay türev alabilmektir.



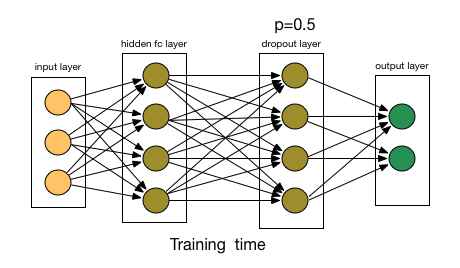
Şekil 3: En bilinen aktivasyon fonksiyonlarının grafikleri

Aktivasyon fonksiyonlarının bazı özellikleri aşağıdaki gibidir;

* İleri beslemeli (Feedforward) ağlarda genelde ReLu versiyonları kullanılmaktadır.
* PeRelu, ELU, Maxout’da kullanılması tavsiye edilen diğer aktivasyon fonksiyonlarıdır.
* Tanjant hiperbolik de kullanılabilir fakat çok fazla bir şey beklememek gerekiyor.
* Sigmoid kullanılması ise genelde tavsiye edilmiyor.
* Tanjant hiberbolik sigmoid göre tercih ediliyor.
* Google tarafından son dönemde ortaya atılan swish fonksiyonu da çok küçükte olsa başarımı bir miktar artırdığı belirtilmektedir.
* Swish, ReLu’ya göre %20 daha yavaş çalışmaktadır. Bununla birlikte 0.001'lik oranında daha başarılı sonuçlar vermektedir.

**Seyreltme (Dropout) Değeri ve Seyreltme Uygulanacak Katmanlar**

Tam bağlı (Fulyy connected) katmanlarda belli eşik değerin altındaki düğümlerin seyreltilmesinin başarımı arttırdığı gözlenmiştir. Yani zayıf bilgilerin unutulması öğrenimi arttırdığı gözlenmiştir.



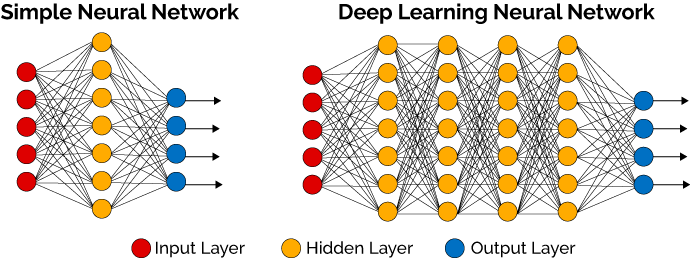
Şekil 4: Örnek dropout uygulaması (Resim kaynak :[Cristina Scheau](https://chatbotslife.com/regularization-in-deep-learning-f649a45d6e0))

Seyreltme (Dropout) değerinin bazı özellikleri aşağıdaki gibidir;

* Seyreltme (Dropout) değeri olarak genelde 0.5 kullanılmaktadır. Farklı kullanıldığı durumlar da yaygındır. Probleme ve verisetine göre değişiklik göstermektedir.
* Seyreltme (Dropout) için rastgele eleme yöntemi de kullanılabilir.
* Seyreltme (Dropout) değeri eşik değeri olarak kullanıldığında [0,1] aralığında bir değer olarak tanımlanmaktadır.
* Tüm katmanlarda aynı dropout değeri kullanılması zorunlu değildir; farklı seyreltme (Dropout) değerleri de kullanılabilir.

**Katman Sayısı ve Gizli Katmanlardaki Nöron Sayısı**

Derin öğrenme yöntemini diğer yapay sinir ağlarından ayıran özellikle de karmaşık problemlerde iyi sonuç vermesini sağlayan en önemli özelliği katman sayısı. Derinlik kavramı da buradan geliyor zaten. Katman sayısı arttıkça modelin daha iyi öğrendiği gözlenmiş. Bildiğim kadarıyla bunun neden böyle olduğu “**matematiksel”** olarak henüz ispatlanmadı(literaür o kadar hızlı ilerliyor ki, bugün çıkmış olabilir:). Ama neden olduğuna dair öngörülerimiz var. Örneğin resim tanıma problemlerinde modelin başlarında elde edilen görüntülerin incelenmesi ile çok katmanlı yapay sinir ağlarında başlarda veriye ait genel hatların öğrenildiği daha sonraki katmalarda gittikçe verinin yavaş yavaş diğer özelliklerinin öğrenildiği gözlendi.



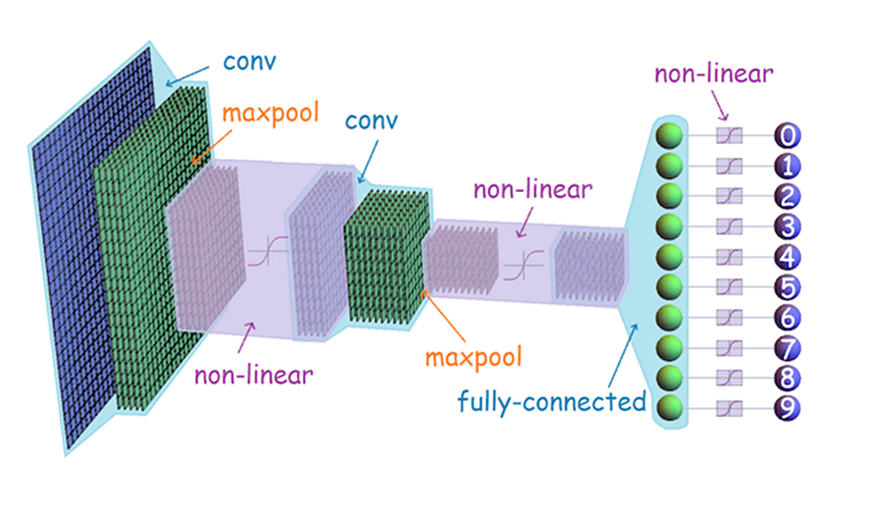
Şekil 7: Örnek “Shallow” ve “Deep” yapay sinir ağı (Resim kaynak :[Abhishek Parbhakar](https://www.linkedin.com/pulse/intuition-deep-neural-networks-abhishek-parbhakar/))

**Peki katman sayısı ne olacak?**

Bunun için elimizde genel geçer bir yöntem yok. Bununla birlikte bazı prensiplerimiz var. Örneğin Andrew Ng’nin söylediği gibi başlarda tek katmanlı veya iki katmanlı bir ağ ile başlayıp uygun hiper parametreler bulunması üzerine yoğunlaşılabilir. Daha sonra gittikçe katman sayısı artırılabilir. İkinci bir durumda katman sayısı arttıkça modelin yapısını da değiştirmek zorunda kalacak olmamızdır. Bunu şöyle bir analojiyle anlatabiliriz. Günümüz de ev yapmaya kalktığımızda 10–15 kata kadar çıkabiliyoruz. Bununla birlikte daha fazla kat çıkmak istediğimizde binamızın temeli bunu kaldıramayacağı için daha güçlü çok katlı bir binaya uygun bir temel üzerine binamızı inşa ederiz. Çok katmanlı yapay sinir ağlarında da benzer şekilde çok katmanlı bir yapı kurulacaksa modelin temeli buna uygun olarak yeniden tasarlanması gerekiyor. Çünkü katman sayısı arttıkça geri basleme (backprobagation) etkisi ilk katmanlara daha az ulaşabilecek. Katman sayısının artırılması belli bir noktadan sonra çok fazla etki etmemeye başlıyor. Bu noktadan sonra diğer hiper parametreler ile modelin başarımı artırılmaya çalışılmalıdır. Bununla birlikte modelin sınırlarını gözlemek bakımında katman sınırı araştırılmalıdır.

Nöron sayısı hafızada tutulan bilgi sayısını gösteriyor. Nöron sayısının fazla olması bellek ihtiyacı ve hesaplama zamanını artırıyor. GPU olmayan bir ortamda çalışıyorsanız bu kriterleri dikkate almanız gerekiyor. Bununla birlikte nöron sayısının az olması yetersiz uyuma (underfitting’e) neden oluyor. Bir diğer özellik de nöron sayısını katmanlar arasında farklı dağılması; yani ilk katmanlarda fazla nöron kullanılırken ilerleyen katmanlarda azlatılırak devam edilmesi “regularization” (başarım iyileştirme) etkisi yaratıyor. Bunun için dropout, sparse-dense-sparse yöntemleri kullanılmaktadır.

**Evrişimli Sinir Ağı (CNN) Kernel Boyutu**



Şekil 8: Örnek Evrişimsel Sinir Ağı (CNN) yapısı (Resim kaynak : [Cornelius Roemer](https://cp4space.wordpress.com/page/2/))

Evrişimsel sinir ağlarında her bir katmanda matris üzerinde işlem yapan kerneller kullanılmaktadır. Bunlar basit Gabor filtreleri gibidir. Bu kernellerin boyutu öğrenme üzerinde hayli etkilidir. Çünkü kernel boyutu ile ne kadar genişlikte verinin birbirini etkileyeceğine karar verilmektedir. Genelde 3x3, 5x5, 7x7 kerneller kullanılmaktadır. Büyük boyutlu kernel kullanılması evrişim uygulandıktan sonra oluşacak resmin küçük olmasına neden olacaktır. Bu durumda bilgi kaybına yol açtığından genelde 3x3 gibi küçük boyutlu kernel kullanılmaktadır. Kenar bulma işleminde işlem yapılan pikselin sağına-soluna üstüne-altına bakılabilmesi için genelde tek sayılardan oluşan merkezi olabilecek filtreler kullanılır.

**Evrişimli Sinir Ağı (CNN) “Pooling” yöntemi: max, min, mean**

Evrişimsel sinir ağlarında kullanılan kernellerin çıktısı üzerinde “pooling” işlemi yapılmaktadır. “Pooling” işlemi kernelde bulunan veriler için filtreleme tekniğidir. Sıkça kullanılan pooling yöntemleri; kernel içindeki matris’de bulunanların en büyüğünü alma “max pooling”, en küçüğünü alma “min pooling” ve ortalamasını alma “mean pooling”dir. Bunlar arasında en sık tercih edileni ve en başarılı sonuç verdiği iddia edileni “max pooling”dir.