

# Week-2

## Error Analysis

### Carrying Out Error Analysis


Geliştirmeye çalıştığınız algoritma hala insan düzeyinde değilse, o zaman algoritmanızın yaptığı hataları manuel olarak incelemek bir sonraki adımda ne yapacağınız konusunda size fikir verebilir. Bu sürece hata analizi denir.

Kedi sınıflandırıcı üzerinde çalıştığınızı ve dev sette %90 acc veya eşdeğerde %10 hata elde ettiğinizi varsayalım ve diyelim ki bu yapmayı umduğunuzdan çok daha kötü. Belki de takım arkadaşlarınızdan biri algoritmanın yanlış sınıflandırdığı bazı örnekler bakar ve bazı köpekleri kediler olarak yanlış kategorize ettiğinizi fark eder ve eğer bu iki köpeğe bakarsanız, belki de en azından ilk bakışta kediye benziyorlar. Bu yüzden, takım arkadaşınızın algoritmanın nasıl daha iyi, özellikle köpeklerde daha iyi çalışacağına dair bir teklifle size gelebilir, değil mi? Belki daha fazla köpek fotoğrafı toplamak, ya da belki köpeklere özel bir şeyler tasarlamak üzere odaklanma çabası yaratmayı hayal edebilirsiniz. Böylelikle kedi sınıflandırıcınızın köpekler üzerinde daha iyi olmasını sağlar ve bu köpekleri yanlışlıkla kediler olarak tanımasını engellersiniz. Öyleyse soru şu ki, devam etmeli ve köpek problemine odaklanmaya başlamalı mısınız? Algoritmanızın köpek resimlerinde çok az hata ile çalışmasını sağlamak için birkaç ay çalışmanız gerekebilir. Peki bu çabalarınıza değer mi? İşe yarayıp yaramayacağını bilmediğiniz için, riski almak için birkaç ay harcamak yerine, çabanıza değip değmeyeceğini çabucak anlamanızı sağlayan bir hata analizi prosedürü var. İşte tavsiye edilen şey;

Öncelikle, 100 tane yanlış etiketlenmiş dev set örneği alın, sonra bunları manuel olarak inceleyin. Bu dev setinizdeki yanlış etiketlenmiş örneklerin kaç tanesinin aslında köpek resimleri olduğunu görmek için her defasında birer birer sayın. 100 örneğin, %5'inin köpek resimleri olduğunu varsayın. Yani, eğer bu yanlış etiketlenmiş dev seti örneklerinden 5'i köpek ise, bunun anlamı 100 örneğini yanlış anlamanızdır. Köpek problemini tamamen çözdüyseniz bile, sadece 100'ün 5'ini daha doğru elde edebilirsiniz. Eğer hatalarınızın sadece %5'i köpek resimleri ise yapabileceğiniz en iyi şey, köpek problemine çok fazla zaman harcadığınızı takdirde, hatanız %10 hatadan %9.5 hataya düşebilir ve bu şekilde zamanınızı en iyi şekilde kullanmadığınıza kanaat getirebilirsiniz. Köpek problemini çözerek iyileştirebileceğiniz maksimum sınırı görmüş olursunuz.

Başka bir örnek olarak 100 yanlış etiket verisinin 50sinin aslında köpek resimleri olduğunu görüyorsunuz. Artık köpek sorununa zaman ayırma konusunda daha iyimser olabilirsiniz. Bu durumda eğer köpek problemini gerçekten çözüyorsanız, hatanızın %10'dan %'5'e düşmesini gözlemleyebilirsiniz ve hatanızın yarıya indirilmesinin çok çaba gerektireceğini görebilirsiniz. Yanlış etiketlenmiş köpeklerin sorununu azaltmaya odaklanın.

Look at dev examples to evaluate ideas



90% accuracy  
→ 10% error

Should you try to make your cat classifier do better on dogs? ←

Error analysis:

- Get ~100 mislabeled dev set examples.
- Count up how many are dogs.

5/100 10% 95% 50% 50/100 10% 5%

Andrew Ng

**hata analizini kullanarak açıklama yapacağız.**

Bazen paralel olarak birden fazla fikri hata analizi yaparak değerlendirebilirsiniz. Örneğin, kedi dedektörünüzü geliştirmek için birkaç fikriniz olduğunu varsayalım.

Belki köpekler üzerindeki performansı artırabilirsiniz?

Ya da büyük kedileri, küçük kediler olarak kabul edildiğini fark edebilirsiniz.

Ya da belki bazı resimlerinin bulanık olduğunu fark ediyorsunuz ve bulanık görüntülerde daha iyi çalışan bir şey tasarlıyorsanız iyi olur.

Bu üç düşünceyi değerlendirmek için hata analizi yaparsak, yapacağımız şey böyle bir tablo oluşturmaktır. Sol tarafta manuel olarak bakmayı düşündüğünüz görüntüler kümesinden geçer. Yani 100 resme bakarsanız bu, 1'den 100'e kadar gider. Sütunlar ise, değerlendirmekte olduğunuz fikirlere karşılık gelecektir. Hata analizi yaparken algoritmanızın yanlış tanıdığı dev set örneklerine bakıyorsunuz. Eğer ilk yanlış tanıyan görüntünün bir köpeğin resmi olduğunu görürseniz, onay işareti koyarsınız.

Son olarak, bir düz görüntüden geçtikten sonra, bu algoritmanızın yüzde kaçının veya bu hata kategorilerinin her birinin yüzde kaçının sütunlara atfedildiğine bakarız. Bazen

diğer hata kategorileri görme durumunuz olabilir. Sürecinizde yeni bir sütun eklemenizin bir sakıncası yoktur.

Bu sürecin sonucu size, bu farklı hata kategorilerinin her biri üzerinde çalışmanın ne kadar değerli olabileceğine dair bir tahmin sunar. Örneğin, msela bu örnekte, bulanık görüntüler üzerinde birçok hata mevcut ve büyük kedilerde. Bu analizin sonucu, bulanık görüntüler üzerinde çalışmanız gerektiği değildir size bir formül vermez fakat takip etmeniz için size en iyi seçenekleri sunar.

Bu nedenle; büyük kedilerdeki, bulanık görüntülerdeki performansı artırmak için ne kadar fikriniz olduğuna bağlı olarak, belki ikisinden birini seçebilirsiniz, ya da takımınızda yeterli personel varsa, görev dağılımı yapabilirsiniz.

**Evaluate multiple ideas in parallel**

Ideas for cat detection:

- Fix pictures of dogs being recognized as cats ←
- Fix great cats (lions, panthers, etc..) being misrecognized ←
- Improve performance on blurry images ←

| Image      | Dog | Great Cats | Blurry | Isolated | Comments        |
|------------|-----|------------|--------|----------|-----------------|
| 1          | ✓   |            |        | ✓        | Pitbull         |
| 2          |     |            | ✓      | ✓        |                 |
| 3          |     | ✓          | ✓      |          | Ring dog at 300 |
| ...        | ... | ...        | ...    | ...      |                 |
| % of total | 8%  | 63%        | 61%    | 12%      |                 |

### Cleaning Up Incorrectly Labeled Data


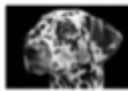





Denetimli öğrenme probleminiz için olan veriler girdi X ve çıktı Y etiketlerinden oluşan verilerdir. Peki verileriniz üzerinden geçerseniz ve bazı çıktı etiketleri Y'lerin yanlış olduğunu, yanlış etikete sahip olduğunu fark ederseniz ne olur? Verinin içerisine bakıp bu hatalı etiketleri düzeltmeniz için harcadığınız zamana değer mi?

Kedi sınıflandırma probleminde, Y eşittir 1 kedi olanlar için, Y eşittir 0 kedi olmayanlar içindir. Dolayısıyla, diyelim ki verilere bakıyorsunuz ve hatalı etiket görüyorsunuz. Hatalı etiketlenme terimini öğrenme algoritmanız hatalı sonuç ürettiğinde bunu adlandırmak için kullanırız. (Algoritmanın ürettiği yanlış etiket değil, veri seti üzerindeki etiketlerden

bahsediyoruz). Buradaki köpek ve Y gerçekte 0 değeri olmalıydı. Fakat belki de bunu etikleyen kişi yanlış etiketledi. Dolayısıyla eğer verilerinizin yanlış etiketlenmiş örneklerle sahip olduğunuzu bulursanız, bu durumda ne yapmalısınız? İlk olarak train setini düşünelim, görünecek olan o ki, derin öğrenme algoritmaları eğitim setindeki rastgele hatalara karşı oldukça dayanıklıdır. Dolayısıyla yanlış etiketlenmiş örneklerinizin rastgeleden çok fazla uzak olmadığı sürece, yani hatalar çoğunlukla rastgele ise, bu durumda hataları olduğu gibi bırakmak büyük ihtimalle kötü bir sonuç doğurmayacaktır, ve hataları düzeltmekle uğraşmak için zaman harcamak gereksiz olacaktır. Tabi ki veri setinize gidip etiketleri inceleyip düzeltmekte sakınca yoktur. Bazen bu yapılmaya değer fakat eğer toplam veri seti oldukça büyükse ve yüzdelik oran çok fazla değilse bunu yapmasanız bile bu çoğu zaman büyük bir hataya sebebiyet vermez. Eğitim setine birkaç X hatası olmasına rağmen eğitimde kullanılan makine öğrenimi algoritmaları gördük ve bunlar çoğunlukla düzgün çalışmakta. Burada bir uyarıya değinmekte fayda var ki bu da, derin öğrenme algoritmaları sistematik hatalara karşı daha az dayanıklıdır fakat rastgele hatalara karşı oldukça dayanıklıdır. Bu dersin konusu yanlış etiketlenmiş örnekler hakkında ne yapılacağı ile ilgili. Dev veya test setindeki yanlış etiketlenmiş verilerle ilgili ne yapmalı?

---

### Incorrectly labeled examples

|   |   |   |   |   |  |   |   |
|---|---|---|---|---|--|---|---|
| x |  |  |  |  |  |  |  |
| y | <u>1</u>  | <u>0</u>  | <u>1</u>  | <u>1</u>  | <u>0</u>   | <u>1</u>  | 1   |

*Training set*

DL algorithms are quite robust to random errors in the training set.

*Systematic errors*

**test setindeki etkisi üzerine endişeleriniz varsa,**

---

Andrew Ng

Önceden oluşturduğumuz tabloya ek bir sütun eklenebilir, burada elle çizilmiş bir kedinin 0'a eşitlendiği ve siz bire eşitlenmesini istiyordunuz. Bu %6'lık hatayı gidip yerinde düzeltmeye gerek var mı? Öneri şudur ki, eğer dev setindeki değerlendirmede önemli

ölçüde fark oluştuyorsa, zaman harcanması gerekir. Fakat sınıflandırıcınızı yüksek miktarda etikleyecek kadar önemli bir fark oluşturmuyor ise uğraşmayın. Fakat uğraşmanızı gerektireceğinizi anlamak için 3 faktör mevcuttur.

Dev set hatası(%10), yanlış etiketlerin hata yüzdesi, ve diğer hatalara bakmalısınız. Bu durumda %9.4'lük kısma odaklanmalısınız.

Başka bir örnek için dev set error'u %2ye kadar düşürdüğünüzü varsayalım, fakat hala %0.6 yanlış etiketlenmeden kaynaklanıyor ve diğer hatalara %1.4 kalır. Bu durumda hata oranının büyük bir kısmı yanlış etiketten olduğundan gidip bu hataları düzeltmeniz çok daha fazla anlam ifade eder.

| Error analysis              |        |           |        |                     |                                   |
|-----------------------------|--------|-----------|--------|---------------------|-----------------------------------|
| Image                       | Dog    | Great Cat | Blurry | Incorrectly labeled | Comments                          |
| ...                         |        |           |        |                     |                                   |
| 98                          |        |           |        | ✓                   | Labeler missed cat in background  |
| 99                          |        | ✓         |        |                     |                                   |
| 100                         |        |           |        | ✓                   | Drawing of a cat; Not a real cat. |
| % of total                  | 8%     | 43%       | 61%    | 6%                  |                                   |
| Overall dev set error       | 10%    |           |        |                     |                                   |
| Errors due incorrect labels | 0.6% ← |           |        |                     |                                   |
| Errors due to other causes  | 9.4% ← |           |        |                     |                                   |

Handwritten calculations and notes:

- 2% (from 6% incorrectly labeled)
- 0.6% (from 2%)
- 1.4% (from 0.6%)
- 2.1% (from 1.4%)
- 1.9% (from 2.1%)

Etiketleri elle düzeltmeye karar verirseniz, burada aklınızda bulundurabileceğiniz birkaç kılavuz ve prensip mevcuttur.

Birincisi, uygulayacağınız işlemin aynı zamanda hem dev, hemde test setine uygulamanız gerektiği olacaktır. Dev ve test setinin aynı dağılımda olması gerektiğini göz önünde bulundurursak bu işlemi iki veri setinede yapmamız gerekmektedir.

İkinci olarak, algoritmanızın hem doğru hemde yanlış olarak belirlediği örnekleri incelemeyi düşünmenizi öneririz. Algoritmanızın yanlış olarak belirlediği örnekleri bakıp bunlardan biri düzeltilmesi gerekiyor mu diye incelemeniz kolaydır. Fakat aynı zamanda algoritmanızın doğru olarak belirlediği fakat hala düzeltilmesi gereken örneklerde mevcut olabilir.

Son olarak, eğer dev ve test verilerinizdeki bazı etiketleri düzeltmeye giderseniz, train setine düzeltme yapmak değişkenlik gösterebilir. Çünkü train seti oldukça büyüktür ve çabanıza değip değmeyeceğini siz belirlemelisiniz.

## Correcting incorrect dev/test set examples

- Apply same process to your dev and test sets to make sure they continue to come from the same distribution
- Consider examining examples your algorithm got right as well as ones it got wrong.
- Train and dev/test data may now come from slightly different distributions.

Windows'u Etkinleştir

### Speech recognition example

→ • Noisy background

→ • Café noise

→ • Car noise

→ • Accent

→ • Far from

→ • Young

→ • Stutter

→ • ...

Guideline:

Build your first system quickly, then iterate.

sisteminizi nasıl geliştireceğiniz konusunda size yardım etmesi için bunu kullanın.

→ • Set up dev/test set and metric

• Build initial system quickly

• Use Bias/Variance analysis & Error analysis to prioritize next

Andrew Ng

## Mismatched Training and Dev/Test Set

### Training and Testing on Different Distributions

Derin öğrenme algoritmaları çok fazla train verisine ihtiyaç duymaktadır. Derin öğrenme algoritmaları genellikle test verisine yalnızca yeterli miktarda train verisi uygulayabildiğinizde en iyi şekilde çalışacaktır. Fakat dev ve test setinin dağılımlarına dikkat edilmesi gerekmektedir.

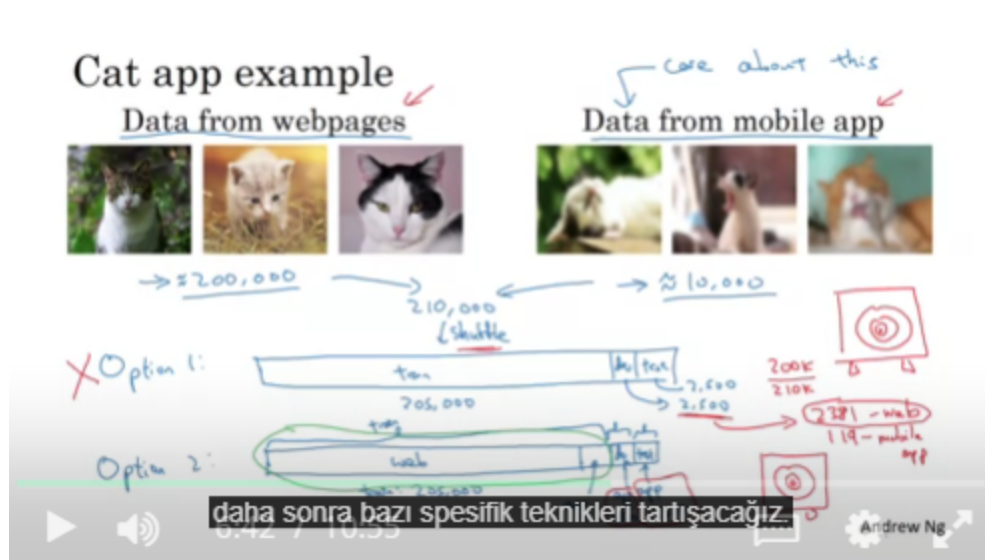
Diyelim ki kullanıcıların cep telefonlarından çekindiği fotoğrafları yükleyebileceği bir mobil uygulama geliştiriyorsunuz ve kullanıcılarınızın mobil uygulamadan yükledikleri fotoğrafların kedi olup olmadığını tanımak istiyorsunuz. Bu durumda, iki veri kaynağı elde edebilirsiniz. Birisi, daha az profesyonelce çekilmiş yani sağdaki gibi mobil uygulamadan gelen, bulanık profesyonel olmayan, sizin gerçekten dikkate aldığınız verinin dağılımıdır. Diğer veri kaynağını elde etme yolu ise internetten arama yapmak ve bu örneğin anlaşılması için farz edelim ki; profesyonelce çerçevelenmiş, yüksek çözünürlükte ve pek çok kedi fotoğrafını indirmektir. Mobil uygulamanız için çok fazla sayıda kullanıcılarınızın olmadığını varsayalım. Bu yüzden, mobil uygulamadan yüklenmiş 10k fotoğraf elde etmiş olabilirsiniz. Ancak, internetten arama yaparak çok fazla sayıda kedi fotoğrafları indirebilirsiniz. İnternetten indirerek 200k kedi fotoğrafı elde etmiş olabilirsiniz.

Bu yüzden dikkate alacağınız şey; son adımınızda, fotoğrafların mobil uygulamada dağılımının üzerinde iyi çalışıp çalışmadığıdır, değil mi? Çünkü en son adımda, kullanıcılarınız sağdaki gibi fotoğrafları yükleyecek ve sınıflayıcınızın bu fotoğraflar üzerinde iyi çalışmasına ihtiyaç duyacaksınız. Ancak şu an küçük bir ikileme düşebilirsiniz. Çünkü, göreceli olarak daha az, dağılımdan çekilmiş yalnızca 10k örnekli, bir veri setine sahip olabilirsiniz. Sizin aslında istediğinizden daha farklı bir fotoğraf görüntüsü bulunmakta. Bu durum göreceli olarak az sayıda train seti almanız ile sonuçlanacağı için sadece 10k fotoğrafı kullanmak istemezsiniz. Bu 200k fotoğrafı kullanmak şimdilik bize yardımcı olacak gibi gözükmemekte. Ancak, bu 200k fotoğrafın sizin tam olarak istediğiniz dağılımdan olmaması bir ikilemdir. Bu durumda ne yapmalıyız?

Bir seçenek şu şekildedir, yapacağınız ilk şey bu iki veri setinin her ikisini de birlikte ele almaktır. Bundan dolayı hali hazırda 210k fotoğrafınız olacaktır. Daha sonra, bu 210k fotoğrafı ele alabilir ve bu fotoğrafları train/dev/test olarak rastgele dağıtabilirsiniz. Farz edelim ki; dev ve test setlerinizin her birinde 2500 örnek yer alacak olsun. Demek ki, 205k train setiniz olacaktır. Bu durumda, verinizin bu şekilde düzenlemenin bazı avantajları olduğu gibi bazı dezavantajları da olacaktır. Veri setlerinizin aynı dağılımda olması sizin için avantaj olacaktır. Ancak büyük bir dezavantajı vardır; 2500 örnekten oluşan dev setinize baktığınızda, asıl dikkate aldığınız olan fotoğrafların mobil



uygulaması dağılımdan ziyade, bunun büyük bir kısmının web sayfalarından gelecek verilerin olmasıdır. Bunun sonucunda, web sayfasından gelen verinizin toplamı 200k olacaktır. Beklenildiği üzere bu 2500 fotoğrafın yaklaşık 2381 tanesi web sayfasından gelecektir. Bu yüzden dev setinizi ayarlamamanın takımınızın nereye hedef olarak yöneleceğini söylediğinizi unutmayın! Birinci seçenek kesinlikle tavsiye edilmeyen durumdur. Bu yüzden bunu yapmak yerine, Eğitim seti, farzedelim ki halen 205k fotoğraf, eğitim setini, internetten alınan tüm 200k fotoğraf olarak alırdım. Daha sonra, eğer istersen, telefon uygulamasından alınan 5k fotoğrafı da ekleyebilirsiniz. Daha sonra dev ve test setiniz için tüm telefon uygulaması fotoğrafları olabilir. Yani dev seti, telefon uygulamasından gelen 2500 fotoğraf, test setide telefon uygulamasının halen 2500 fotoğraf olacaktır. Bu şekilde verinizi train, dev ve test olarak böleminin avantajı, olmasını istediğiniz gibi hedefe yönelmiş olmanızdır. Tabi ki dezavantaj olarak; hali hazırda train veri setinizin dağılımı dev ve test setlerinin dağılımlarından farklıdır. Ancak, verinizi eğitim, dev ve test olarak bölmeniz uzun vadede daha iyi performans almanızı sağlar. Dev ve test setlerinin dağılımlarından farklı dağılıma sahip train setleriniz ile baş etmeniz için daha sonra bazı spesifik teknikleri tartışacağız.



Yeni edelim ki; yeni bir ürün markası geliştiriyorsunuz, bir araba için aktif konuşan bir dikiz aynası.



## Speech recognition example

Speech activated rearview mirror

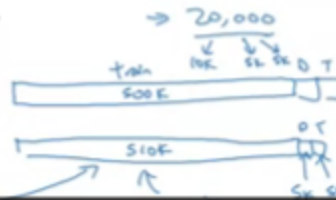


### Training

Purchased data  $X, y$   
Smart speaker control  
Voice keyboard

### Dev/test

Speech activated  
rearview mirror



Bu sana, çok daha büyük eğitim seti, 500,000 sesten daha fazla, verir.

## Bias and Variance with Mismatched Data Distributions

Öğrenme algoritmanızın bias ve varyansını tahmin etmek, bir sonraki aşamada ne yapacağınızda öncelik vermenizde yardımcı olur. Ancak, train setiniz, dev ve test setlerinizden farklı bir dağılımdan geldiğinde, bias ve varyansı analiz etme şekliniz değişir.

Kedi sınıflandırma örneğimizi kullanmaya devam edelim ve diyelim ki insanlar bunun üzerinde mükemmele yakın bir performans sergiliyorlar. Yani Bayes hatası bu problem için yaklaşık %0 olduğunu biliyoruz. Bu yüzden, hata analizi yapmak için genellikle train hatasına bakarsınız ve ayrıca dev setindeki hataya bakarsınız. Öyleyse, bu örnekte eğitimin hatasının %1 olduğunu ve dev hatasının %10 olduğunu söyleyelim. Eğer dev setinizin, train setiniz gibi aynı dağılımdan gelseydi, burada algoritmanızı daha önceden çok daha kötüye giden dev setini daha iyiye götüren eğitim setinden genelleme yapmadığından büyük bir varyans probleminiz olduğunu söyleyebilirsiniz. Ancak, train setinizin ve dev setinizin farklı bir dağılımdan geldiği ortamda, artık bu sonucu güvenli bir şekilde çizemezsiniz. Eğer train seti çok iyi verilerden fakat dev setiniz bozuk görüntülerden oluşuyorsa belki bir varyans probleminiz yoktur ve bu sadece dev setinizin doğru sınıflandırılması çok daha zor olan görüntüler içerdiğini yansıtır. Yani bu analizdeki problem, eğitim hatasından dev hatasına gittiğinizde, her seferinde iki şeyin değişmesidir. Birincisi, algoritmanın train setindeki verileri gösterdiği, ancak dev setinde bulunmadığıdır. İkincisi, dev setindeki verilerin dağılımı farklıdır ve aynı anda iki şeyi değiştirdinizden dolayı, bu %9luk hata artışını bilmek zordur, ne kadarı algoritmanın dev

setinde görmediğinden dolayı, bu problemin varyans bölümünün bir kısmı. Çünkü dağılımlar farklıdır. Yani, bu iki etkiyi ortaya çıkarmak için, ve bu iki farklı etkinin ne olduğunu tam olarak takip etmediyseniz, endişelenmeyin.

Ancak, bu iki etkiyi ortaya çıkarmak için, train-dev seti olarak adlandıracağımız yeni bir parçası tanımlamak faydalı olacaktır. Yani bu, train setleriyle aynı dağılıma sahip olması gereken, oluşturduğumuz yeni bir veri alt setidir, ancak bu konuda ağınızı açık bir şekilde eğitmiyorsunuz. Yani daha önce bazı train setleri ve bazı dev setleri ve test setleri oluşturmuştuk ve dev ve test setleri aynı dağılıma sahiptir, ancak train setleri biraz farklı dağılıma sahip olacaktır. Yapacağımız şey, train setlerini rastgele karıştırmak ve daha sonra train-dev seti olarak train setinin bir parçasını oluşturmak olacaktır. Yani, dev ve test setinin aynı dağılımı olduğu gibi, train setide train-dev setide aynı dağılıma sahiptir. Ancak, fark şu ki, artık sadece uygun train setinde sinir ağınızı eğitiyorsunuz. (gerçek train setinde). Hata analizi yaparken artık train-dev setinide dahil ediyoruz. Yeni yazdığımız örnekte, train setinden train-dev setine hatanın gerçekten çok arttıdır ve train seti ile train-dev verisi arasındaki tek fark, sinir ağınızın bunun ilk bölümünü sıralamasıdır. Yani train-dev setinin açıkça eğitime dahil olmamasıdır. Yani bu size varyans probleminiz olduğunu söyleyebilir. Çünkü train-dev hatası, train setiniz gibi aynı dağılımdan gelen veriler üzerinde ölçülmüştür. Bu nedenle, sinir ağınızın bir train setinde iyi bir performans göstermesine rağmen, aynı dağılımdan gelen train-dev setinde varyans problemi yaratıyor çünkü train-dev seti genelleştirmek için train edilmedi.

Farklı bir örneğe bakalım, train hatasının %1 ve train-dev hatasının %1.5 olduğunu, ancak dev setine gittiğinizde hatanızın %10 olduğunu görüyorsunuz. oldukça düşük bir varyans probleminiz var, fakat dev setine nazaran büyük bir hata gözüktüğünden verilen uyuşmadığı bir sorun mevcut. (data mismatched). Çünkü öğrenme algoritmanız train-dev ya da dev setinden açıkça eğitilmemişti, bu iki veri seti farklı dağılımdan geliyor. Ancak öğrendiği algoritma ne olursa olsun, train-dev seti üzerinde harika çalışıyor, ancak dev seti üzerinde iyi çalışmaz. Öyleyse, algoritmanızın bir şekilde gerçekten önemseydiğinizde farklı bir dağılım üzerinde öğrenildiğinden dev setinde sıkıntı çıkartır ve buna data mismatched diyoruz.

Train %10, train-dev %11, dev %12 olarak belirlensin ve bayes hatası için %0 varsayalım. Öyleyse bu durumda önlenabilir düzeyde bir yüksek bias sorunuyla karşı karşıyasınızdır. Çünkü insan seviyesinden çok daha kötü bir performans sergiliyorsunuz.

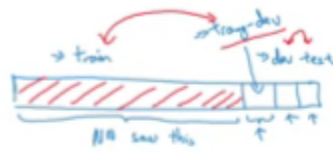
## Cat classifier example

Assume humans get  $\approx 0\%$  error.

Training error .....  $1\%$

Dev error .....  $10\%$

Training-dev set: Same distribution as training set, but not used for training



|                      |        |                             |         |                             |
|----------------------|--------|-----------------------------|---------|-----------------------------|
| Training error       | $1\%$  | $\downarrow$ variance       | $1\%$   | $\downarrow$ variance       |
| → Training-dev error | $9\%$  |                             | $1.5\%$ | $\downarrow$ data mismatch  |
| → Dev error          | $10\%$ |                             | $10\%$  |                             |
| Variance             |        |                             |         |                             |
| Human error          | $0\%$  | $\downarrow$ avoidable bias | $10\%$  | $\downarrow$ avoidable bias |
| Training error       | $10\%$ | $\downarrow$ variance       | $10\%$  | $\downarrow$ variance       |
| Training-dev error   | $11\%$ |                             | $11\%$  | $\downarrow$ data mismatch  |
| Dev error            | $12\%$ |                             | $20\%$  |                             |

büyük bir yanlışlık veya önlenabilir yanlışlık probleminiz olduğunu söyleyeceğim.

Andrew Ng

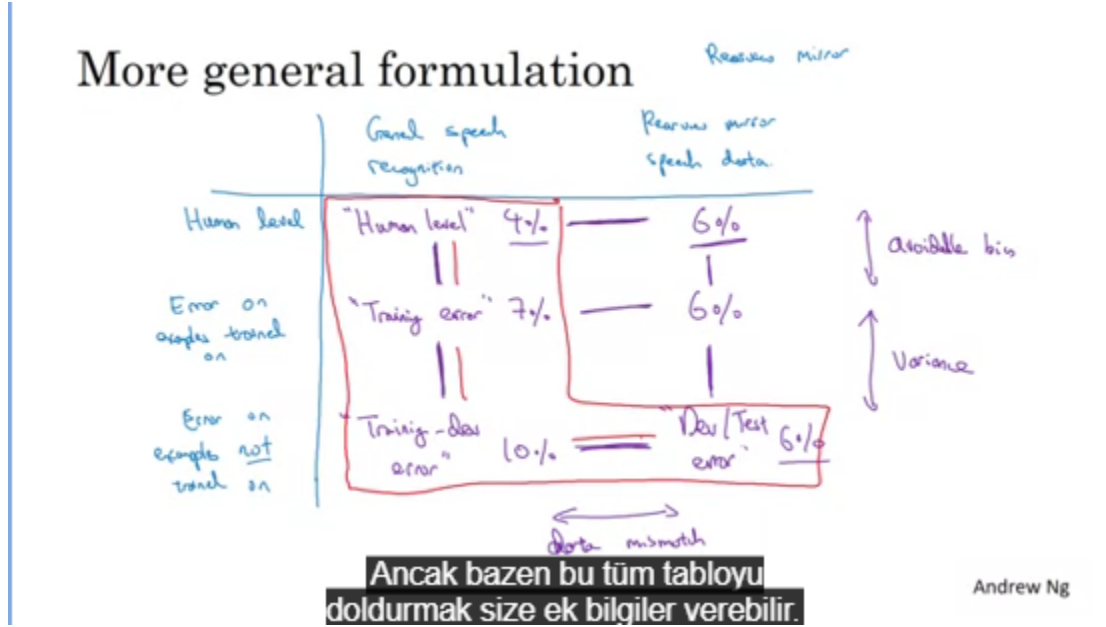
Bu slaytta yaptığımız şeyi alıp genel ilekeri yazalım. Bakacağımız anahtar nicelikler insan seviyesi hatası, train set error, train-dev error(train ile aynı dağılım fakat eğitilmedi), dev set error ve bu hatalar arasındaki farklara bağlı olarak bias varyans ve data mismatch. Dev seti ve test seti arasında fark varsa bu overfitting anlamına gelir. Belkide daha büyük bir dev seti bulmalısınız. Bu durumda dev ve test setinizin aynı dağılımdan geldiğini unutmayın. Sağdaki durumda train ve train-dev fit ederken kullanıldı. Bunu daha iyi anlayabilmek için genel bir formalizasyona göz atalım.

## Bias/variance on mismatched training and dev/test sets

|                        |        |   |        |
|------------------------|--------|---|--------|
| Human level            | $4\%$  | $\downarrow$ avoidable bias                   | $4\%$  |
| Training set error     | $7\%$  | $\downarrow$ variance                         | $7\%$  |
| Training-dev set error | $10\%$ | $\downarrow$ data mismatch                    | $10\%$ |
| → Dev error            | $12\%$ | $\downarrow$ degree of similarity to dev set. | $6\%$  |
| → Test error           | $12\%$ |   | $6\%$  |

Genel seslerin olduğu ve sadece dikiz aynası konuşma tanımasından aldığınız veriler var.

Satır başlıklarında human-level, fit edilen train data hatası, , fit edilmeyen train-dev örnek hatası,



## Addressing Data Mismatch

Veri uyumsuzluğu(mismatch) sorunu ile nasıl baş edebiliriz?

Büyük bir veri uyumsuzluğu sorunun olduğunu fark edersem, genellikle yaptığım şey manuel hata çözümlemesini gerçekleştirmek ve train seti ile dev/test seti ile arasındaki farkları anlamaya çalışmaktır. Özellikle hata çözümlemesi söz konusu olduğunda test kümesinin overfitting durumundan kaçınmak için manuel olarak test kümesi yerine sadece dev setine bakmalısınız. Ses tanımalı dikiz aynası uygulaması geliştirirken farkı göz önünde bulundurursanız, dev setindeki seslerin gürültülü olduğunu fark edebilirsiniz ve sizin dev setinizin, train setinizden farklı olmasını sağlayan sorunu görebilirsiniz. Belki diğer hatalara göz atabilirsiniz. Mesela, aracınızdaki konuşma ile etkin olan dikiz aynasında genellikle sokak numaralarının yanlış tanımlandığını tespit edebilirsiniz. Bu durumda yapacağınız şey, train verilerini daha benzer hale getirmenin yollarını bulmaya çalışmaktır ya da dev/test kümelerine benzer daha fazla veri toplayabilirsiniz. Örneğin, arka plandaki otomobil gürültüsünün büyük bir hata kaynağı olduğunu görürseniz, yapabileceğiniz tek şey, gürültülü araç içi verileri taklit etmektir. Sokak numaralarını tanımakta zorlanıyorsanız bu durumda gidip sayılardan bahseden insanlardan daha fazla veri almayı deneyebilir ve train setinize ekleyebilirsiniz. Yani, eğer amacınız train setinizi dev setinize daha benzer yaparsa, yapabileceğiniz şeyler nelerdir?

Kullanabileceğiniz tekniklerden biri yapay veri sentezi oluşturmak, araç gürültülü problemini ele alma bağlamında tartışalım.

## Addressing data mismatch

- • Carry out manual error analysis to try to understand difference between training and dev/test sets

E.g. noisy - car noise      street noises

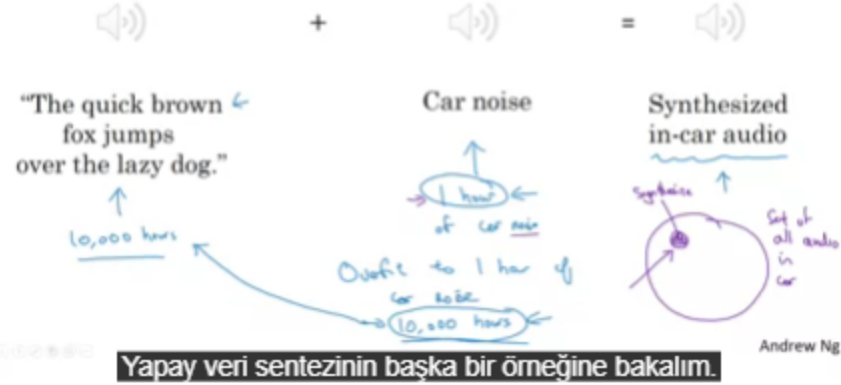
- • Make training data more similar; or collect more data similar to dev/test sets

E.g. Simulate noisy in-car data

Örneğin bir konuşma tanıma sistemi kuruyorsanız, otomobil içinde; bir otomobilin arka plan gürültüsüyle, bir otoyolun arka plan gürültüsüyle vs birlikte, kaydedilmiş çok fazla sesiniz yok belki de. Ancak bunu sentezlemenin bir yolu var. Örneğin, bu otomobil arka plan gürültüsü olmadan çok miktarda temiz ses kaydettiğini varsayalım. Temiz bir ses ile otomobil gürültüsünün kaydı ile sentezleme yapabilirsiniz. Bu temiz veri ile gürültülü bir ortamda söylemek yerine veriyi sentezleyip yapay veri üretebilirsiniz.

Yapay veri sentezi konusunda dikkat edilmesi gereken husus, diyelim ki sessiz bir arka plana karşı kaydedilen 10.000 saat veri var ve diyelim ki sadece 1 saatlik otomobil gürültünüz var. Bu durumda 10k saatli veri için 10bin kere sentez yaparsınız fakat bunu yaparsanız, ses insan kulağına kusursuz bir biçimde gelir ve öğrenme algoritmanızın bir saatlik otomobil gürültüsüne karşı overfitting olma riski vardır.

## Artificial data synthesis



Başka bir örnekte ise yapay veri sentezinin başka bir örneğine bakalım. Diyelim ki sürücüsüz bir otomobil inşa ediyorsunuz ve bu yüzden böyle araçları gerçekten tespit etmek ve etrafına bir sınırlayıcı kutu koymak istediğinizi varsayalım. Pek çok insanın cevabını aradığı soru ise neden tonlarca araç görüntüsünü taklit etmek için bilgisayar grafiklerinin kullanılması gerektiğidir. Tüm araçların olduğu kümede çok az bir alt kümede bu sentezlenmiş görüntüler göze iyi görünür. Ancak sentezlediğiniz bu küçük alt kümenin overfit olmasına sebepte olabilirsiniz. Bir oyuna bağlı (20 araç içeren) bir sinir ağı geliştiriyorsanız bir problem yok fakat gerçek dünya için binlerce araç çeşidi olduğundan 20 otomobile overfit gösterecektir.

## Artificial data synthesis

Car recognition:



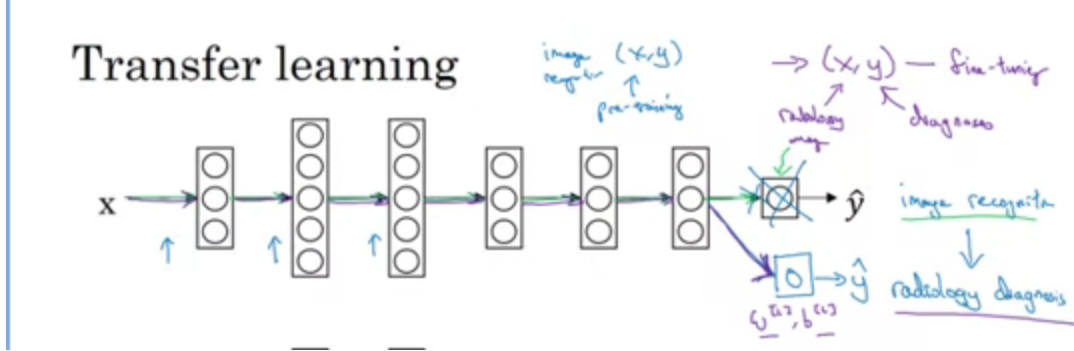
## Transfer Learning

Derin öğrenmedeki en güçlü fikirlerden biri, bazen sinir ağının bir görevden öğrendiği bilgiyi almak ve o bilgiyi ayrı bir göreve uygulayabilmektir. Yani, örneğin, sinir ağınız kediler gibi nesneleri tanımayı öğrenmiş olabilir ve o bilgiyi ya da o bilginin bir parçasını size röntgen taramalarını daha iyi okumanızda yardım etmesi için kullanabilirsiniz.

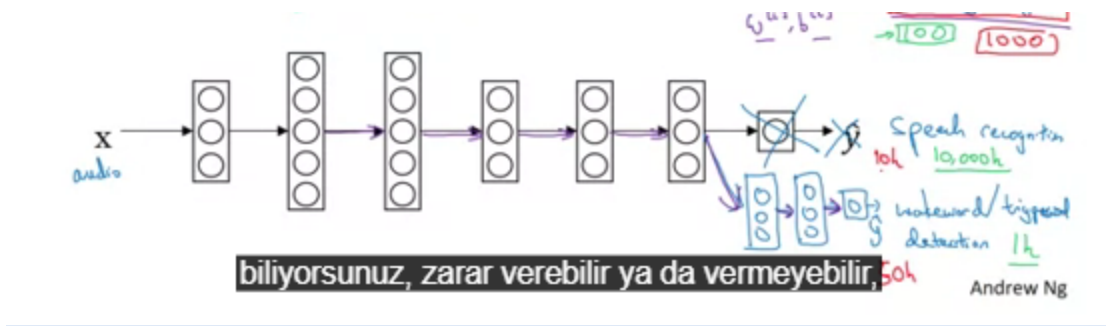
Diyelim ki sinir ağınızı görüntü tanıma üzerine eğitmiş olun. O halde önce sinir ağı alırsınız ve onu X'in bir görüntü ve Y'nin herhangi bir nesne olduğu X Y çiftleri üzerine eğitirsiniz. Görüntü bir kedi veya bir köpek veya bir kuş ya da başka bir şeydir. Eğer bu sinir ağını alır ve öğrenileni uyarlamak isterseniz ya da başka bir göreve transfer demek daha doğru olur, radyoloji tanısı gibi, sinir ağının bu son çıktı katmanını çıkarmak ve aynı zamanda o son çıktı katmanını besleyen ağırlıklarında silmektir ve sadece son katman içi rastgele oluşturulmuş yeni bir ilk değer ağırlıklar dizisi oluşturmalısınız ve bunları yaptıktan sonra radyoloji tanısı ortaya çıkar. Kedi algoritması eğittiğinizde aslında görüntüyü tanıyan iyi bir sinir ağı oluşturmuş olursunuz. Transfer learning'i gerçekleştirmek için şimdi yapmanız gereken, radyoloji görüntülerinden oluşan, yeni bir X Y veri kümesini kullanmaktır. X'ler radyoloji tanıları, Y'ler tahmin etmek istediğiniz tanılar ve yapmanız gereken son katmanların ağırlıklarının ilk değerlerini vermek. Bu yeni veri kümesi ile sinir ağını yeniden eğitelim. Radyoloji verisiyle sinir ağınızı yeniden eğitmek için bir kaç seçeneğiniz var. Eğer küçük bir radyoloji veri kümeniz varsa, sadece son katmanın ağırlıklarını, sadece  $w[b], b[l]$ , yeniden eğitmek isteyebilirsiniz ve diğer değişkenleri sabit tutmak isteyebilirsiniz. Eğer yeterince veriniz varsa, sinir ağının kalan tüm katmanlarını yeniden eğitebilirsiniz ve ana kural olarak eğer küçük bir veri kümeniz varsa o zaman sadece en son katmanı yani çıktı katmanını eğitebilirsiniz. Ya da belki son bir ya da iki katmanı eğitebilirsiniz. Büyük veriniz varsa tüm katmanları baştan eğitebilirsiniz ve eğer sinir ağındaki tüm değişkenleri yeniden eğiterseniz, o zaman görüntü tanıma üzerine olan eğitimin ilk aşaması bazen eğitim öncesi(pre-training) diye adlandırılır. Çünkü görüntü tanıma verilerini sinir ağının ilk ağırlık değerlerini vermede ya da aslında ilk eğitimde kullanıyorsunuz ve daha sonra tüm ağırlıkları güncelliyorsanız, ondan sonra radyoloji verisiyle eğitim yaptığınızda o zaman bazen bu ince ayar(fine-tuning) diye adlandırılır. Bu uygulamada yaptığınız görüntü tanımadan öğrendiğiniz bilgiyi alıp, onu radyoloji tanısına uygulamak ya da aktarmaktır. Bunun faydalı olabilmesinin nedeni, köşeleri algılama, eğrileri algılama, pozitif nesneleri algılama gibi bir çok alt seviye özelliktir. Böyle büyük bir görüntü tanıma veri kümesinden öğrenmek öğrenme algoritmanızın radyoloji tanısında daha iyi olmasına yardım edebilir. Görüntülerin yapısının ve doğasının neye benzediği konusunda bir çok



bilgi öğrenmiş olur ve bu bilginin bir kısmı faydalı olacaktır. Yani görüntüleri tanımayı öğrenerek, farklı görüntülerin bazı parçalarının neye benzediğini öğrenmiş olabilir. Çizgiler, noktalar, eğriler vb.



Diyelim ki bir konuşma tanıma sistemi eğitmişsiniz. O zaman  $X$  ses ya da ses parçacıkları girdisidir, ve  $Y$  bazı yazılı metinlerdir. Yani konuşma tanıma sisteminizi metin çıktısı vermek için eğittiniz. Diyelim ki “uyandırma kelimeleri” ya da “tetikleme kelimeleri” algılama sistemi yapmak istiyorsunuz. O zaman, uyandırma ya da tetikleme kelimelerinin evimizde konuşma kontrollü cihazları uyandırmak istediğimizde söylediğimiz kelimeler olduğunu düşünerek bir cihazı uyandırabilirsiniz. Bunu yapabilmek için sinir ağının son katmanını tekrar çıkarır ve yeni bir çıktı düğümü oluşturabilirsiniz. Ama bazen yapabileceğiniz diğer şey sadece tek bir yeni çıktı yaratmak değil, ama uyandırma kelimesi algılama sorunuz için  $Y$  etiketleri koymaya uğraşırken sinir ağınızda birkaç tane yeni katman yaratmaktır.



Özetlemek gerekirse, ne zaman transfer learning bir anlam ifade eder?

Eğer bir  $A$  görevinden öğrenmeye çalışıyorsanız ve bilginin bir kısmı  $B$  göreve aktarmaya çalışıyorsanız, o zaman aynı girdi  $X$ 'e sahip  $A$  görevinden  $B$  göreve transfer learning bir anlam ifade eder.

## When transfer learning makes sense

Task from A  $\rightarrow$  B

- Task A and B have the same input  $x$ .
- You have a lot more data for Task A than Task B.  
 $\uparrow$   $\uparrow$
- Low level features from A could be helpful for learning B.

### Multi-task Learning

Transfer learningte görev A'dan öğrenip bunu görev B'ye aktardığınız ardışık bir süreciniz varken, multi-task learningte aynı anda başlıyor ve bir sinir ağının aynı anda birden çok şey yapmasını sağlamaya çalışıyorsunuz ve sonra bu görevlerden her biri umuyoruz ki tüm diğer görevlere yardımcı olur.

Diyelim ki, otonom bir araç inşa ediyorsunuz ve bu araç sürücüsüz olacak. O zaman sürücüsüz aracınızın yayalar, diğer araçlar, dur işaretleri gibi birden fazla değişik şeyi tespit etmesi gerekecektir. Yani örneğin, soldaki bu örnekte, bu görselde bir dur işareti var ve bu görselde bir araba var ama trafik ışığı veya yaya yok. Yani eğer bu görsel bir örnek için girdi ise,  $x(i)$ , o zaman bir tane  $y(i)$  etiketi yerine 4 etiketiniz olurdu. Tespit etmek istediğiniz şey artarsa etiketinizde artacaktır.

Şimdi yapabileceğiniz sinir ağınızı  $y$ 'nin bu değerlerini öngörmek için eğitmek.

### Simplified autonomous driving example



Handwritten notes illustrating the output vector  $y^{(i)}$  for the simplified autonomous driving example:

Labels: Pedestrians, Cars, Stop signs, Traffic lights

Output vector  $y^{(i)}$  (size 4,1):

$$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Matrix  $Y$  (size 4,m):

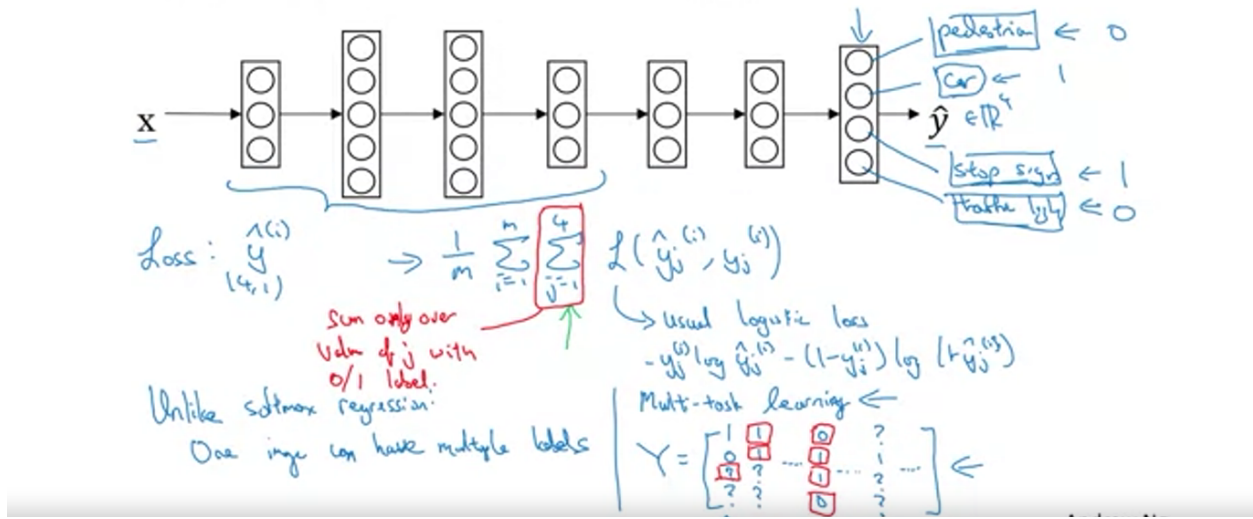
$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & \dots & y^{(m)} \end{bmatrix}$$

Böylece bir sinir ağı girdiniz ve  $x$  ve şimdi dört boyutlu bir değeri olan  $y$  çıktınız olabilir. Dikkat ederseniz burada çıktı için dört düğüm vardır.(output layer)

Sinir ağını eğitmeye başlamak için maliyeti bulmamız gerekiyor. (Logistic Loss). Softmax regresyondan farklı olarak bu bir göselin birden fazla etiketi olabilir. Yani, her bir görselin bir yaya resmi veya araba resmi, stop levhası resmi trafik ışığı resmi olup olmadığını söylemiyorsunuz. Her bir resim için tanımlanan nesnelerin 1-0 değerlerine bakıyorsunuz. Eğer bir sinir ağına maliyet fonksiyonunu en aza indirmek için eğitiyorsanız, multi-task learning uyguluyorsunuz demektir. Çünkü yaptığınız her bir görsele bakan ve basitçe dört sorunu çözen tek bir sinir ağı inşa etmektir. Size her bir görsel için bu dört nesneden her biri var mı bunu söylemeye çalışıyor. Yapabileceğiniz bir şey de, bir sinir ağını dört şey yapmak için eğitmek yerine 4 ayrı sinir ağı eğitmektir. Ama eğer sinir ağındaki daha önceki özelliklerden bazıları bu değişik tipteki nesneler arasında paylaştırılabilirse, o zaman dört tamamen ayrı sinir ağını tek tek eğitebilirsiniz. Daha iyi performans gösterir fakat biraz zordur.

Önemli bir detay olarak, şimdiye kadar bu algoritmayı sanki her görselde her bir etiket varmış gibi tanımladık. Ortaya çıkıyor ki, multi-task learning bazı görsellerde sadece bazı nesneleri etiketlese bile çalışıyor. Diyelim ki biri etiketleyiciniz, orada bir yaya olduğunu, bir araba olmadığını ama bir dur levhası veya trafik ışığı olup olmadığını etiketlemekte uğraşmadıklarını söyledi. Yani soru işareti kalır. İşte buna benzer bir veri kümesi ile halen öğrenme algoritmanızı aynı anda dört görev yapmak için eğitebilirsiniz, bazı görseller etiketlerin sadece bir altkümesine sahip olsa ve bazıları soru işareti veya önemseme içerse de ve algoritmayı eğitme yolunuz, bu etiketlerin bazıları soru işaretleri olsa veya gerçekten etiketlenmemiş olsa da sadece  $j$ 'nin 0 veya 1 etiketine sahip olduğu değerler üzerinden toplanacaktır. Soru işareti olduğu durumlarda soru işaretleri  $J$ 'ye dahil edilmez.

## Neural network architecture



O halde, multi-task learning ne zaman bir anlam ifade ediyor?

Genelde üç şeyin doğru olduğu zamanlarda mantıklı olur.

Birincisi eğer düşük seviyeli özellikleri paylaşıyor olmaktan fayda sağlayacak görevler kümesinde üzerinde eğitiyorsanız. Yani sürücüsüz araç örneği için, trafik ışıkları ve arabalar ve yayaları tanımak mantıklıdır, bunların aynı zamanda dur levhalarını tanımanızı sağlayacak benzer özellikleride olmalıdır. Çünkü tüm bunlar yolların özellikleridir.

İkincisi, bu kural her zaman doğru değildir. Her bir görev için elinizde olan veri miktarı oldukça benzerdir. Yani eğer transfer learningten hatırlarsanız, bir A görevinden öğreniyor ve bir B görevine aktarıyorsunuz, yani eğer A görevinden bir milyon örnek varsa ve B görevi için 1.000 öğrenmek varsa, o zaman bu bir milyon örnekten öğrendiğiniz tüm bilgi B görevi için sahip olduğunuz daha küçük veri kümesini arttırmanıza yardımcı olabilir. Peki multi-task learning? Genellikle ikiden çok daha fazla görev vardır. Diyelim ki 100 göreviniz var ve diyelim ki 100 farklı nesneyi aynı anda tanımlamak için multi-task learning kullanacaksınız, görev başı 1000 örneğe sahip olabilirsiniz ve eğer sadece bir görevin performansına odaklanırsanız, diyelim ki 100. görevin performansına odaklanalım. Eğer bu 100. görevi tek başına yapmaya çalışırsanız, bu örneği eğitmek için sadece 1000 örneğiniz olmuş olur, Diğer 99 görev için 99.000 eğitim örneği anlamına gelir ve bu da önemli bir artış sağlar. Veri artırılabiliyor düzeyde olabilir. Her bir görev için bu kombinasyon geçerlidir.

Son olarak multi-task learning için tüm görevlerde iyi sonuç alabilmek için yeterli büyüklükte sinir ağıyla eğitildiğinde daha fazla anlama sahip olmaktadır.

### When multi-task learning makes sense

- Training on a set of tasks that could benefit from having shared lower-level features.
- Usually: Amount of data you have for each task is quite similar.



- Can train a big enough neural network to do well on all the tasks.

Andrew Ng