COURSE3 – W2

Vid 1

Carrying Out Error Analysis

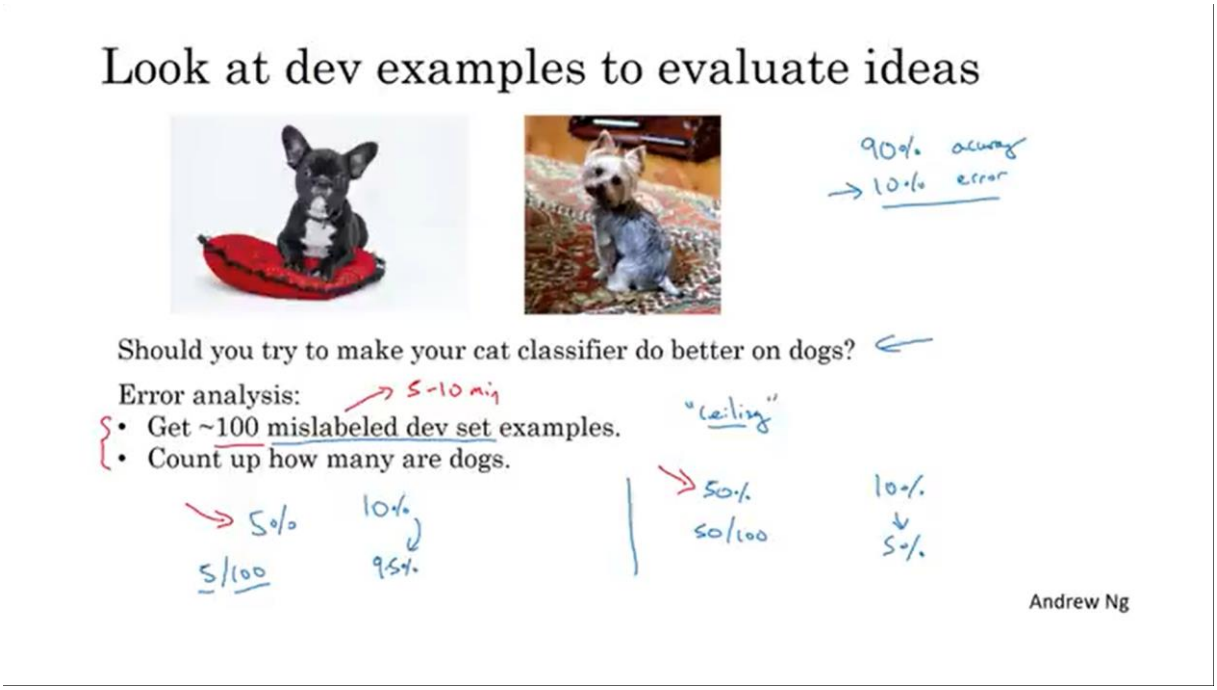
Eğer insanların yapabildiği bir task üzerinde çalışıyorsak ve modelimizin performansı HumanLevelPerformance'ı henüz geçmediyse, o halde modelin hatalı sonuç bulduğu data örneklerini manuel olarak incelemek modeli geliştirmek için bir sonraki adımda ne yapılması gerektiği ile ilgili fikir verebilir. Bu sürece Error Analysis denir.

Bir örnekle konuyu daha iyi anlayalım.

Diyelim ki bir cat classifier eğitiyoruz ve ve dev set üzerinde %90 accuracy sağladık yani %10 error var, bu beklediğimizin çok daha kötü.

Manuel olarak algoritmanın hatalı bulduğu örneklere bakınca görüyoruz ki algoritmanın hatalı sonuç verdiği bazı datalar aşağıdaki gibi köpek dataları, baktığımız zaman aslında bu köpekler kediye bir hayli benziyor.

Bu durumda acaba modelimizi köpek fotoğrafları konusunda daha iyi hale getirmemiz işe yarayabilir mi? Belki daha fazla dog pictures kullanabiliriz veya belki köpeklere özel bazı feature'lar kullanabiliriz. Soru şu, bunu yapmalı mıyız? Gerçekten modelin köpekler üzerindeki performansını artırmak için uğraşmalı mıyız? Belki aylar boyunca buna uğraşacağız ama sonucunda model performansının çok da iyileşmediğini göreceğiz.



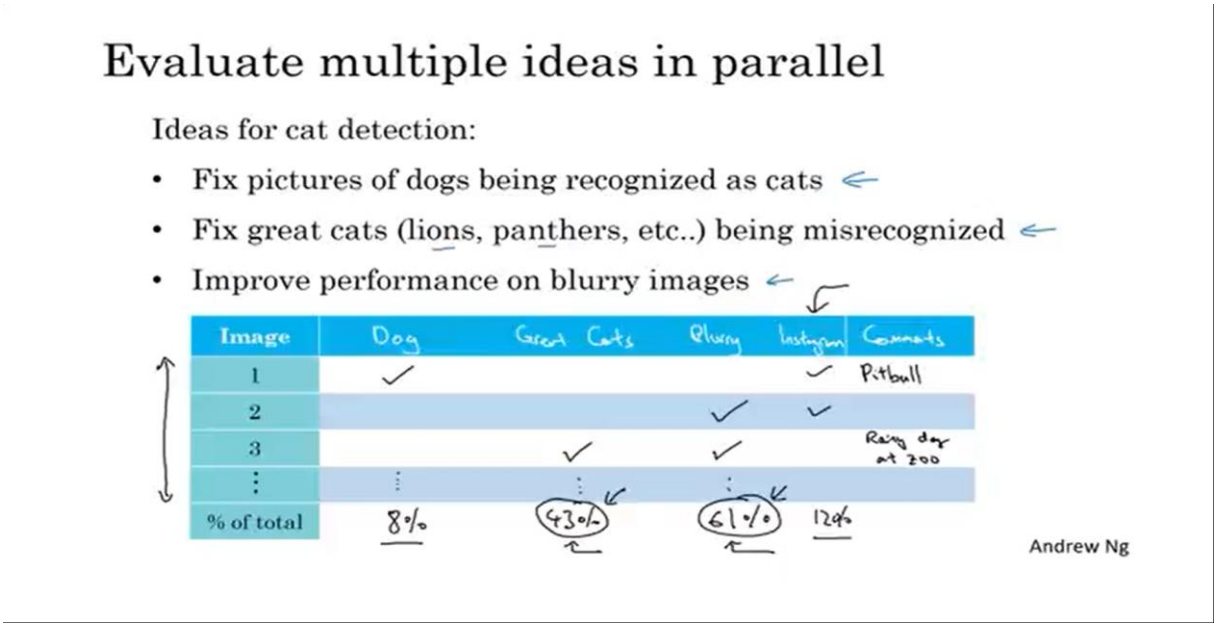
Doğrudan bu işe atılmak yerine önce yukarıdaki gibi bir error analysis procedure izleyip, bunu yapmanın değip değmeyeceğine karar vermek.

Bunu yapmak için basitçe dev setinde hatalı ayırt edilen 100 örneğe bakarız, kaç tanesinin köpek fotoğrafı olduğunu sayarız, diyelim ki 5 tanesi köpek fotoğrafı yani bu şu demek: dev set hatalarının ortalama %5'i köpek fotoğraflarını misclassify etmemizden kaynaklanıyor. Bu durumda bu problemi tamamıyla çözsek dahi train set hatamız %10'dan %9.5'e düşecek. Çabaya değmez.

Buna karşın eğer 100 hatalı ayrımın 50 tanesinin dog pictures olduğunu görseydik bu durumda hatayı %10'dan %5'e çekme şansımız olduğunu biliyoruz, bu durumda modelin köpekler üzerinde performans artırması için çalışma yürütebiliriz.

Yukarıdaki örnekte error analysis ile tek bir sorunun (modelin köpek fotoğraflarını ayırması için daha fazla çalışmalı mıyız?) cevabını bulmaya çalıştık, bazen birden fazla sorunun cevabını paralel şekilde error analysis ile bulmaya çalışabiliriz.

Aşağıda buna bakalım. Diyelim ki cat classifier modelinizi geliştirmek için birden fazla fikriniz var, hatanızın sebebinin köpek fotoğrafları veya great cats olabileceğini düşünüyorsunuz, ya da blurry images yüzünden beklediğinizden düşük performans sergiliyorsunuz. Enerjinizi hangisi üzerine yoğunlaştırmalısınız?



Bunun için yukarıdaki gibi bir tablo yaparız, dikey eksende dev setinde hatalı classify edilen örnek numarası, yatay eksenler ise olası sebepler. Tek tek 100 örneği inceleriz ve sebebini işaretleriz gerekirse not alırız, bazen bu inceleme sırasında yeni olası hata sebepleri farkederiz mesela hataların bir kısmında instagram filtreleri tespit ederiz.

Sonuçta yüzdelere bakarak, hangi probleme yoğunlaşmamız gerektiği ile ilgili bir fikir edinebiliriz, hangi problemi tamamen çözersek bize performans getirisinin ne kadar olacağını aşağı yukarı görmüş oluruz.

Vid 2

Cleaning Up Incorrectly Labelled Data

Bazen dataset içerisinde bazı örnekler yanlış etiketlenmiş olabilir bu durumda nasıl bir yaklaşım izlemeliyiz? Bakalım.

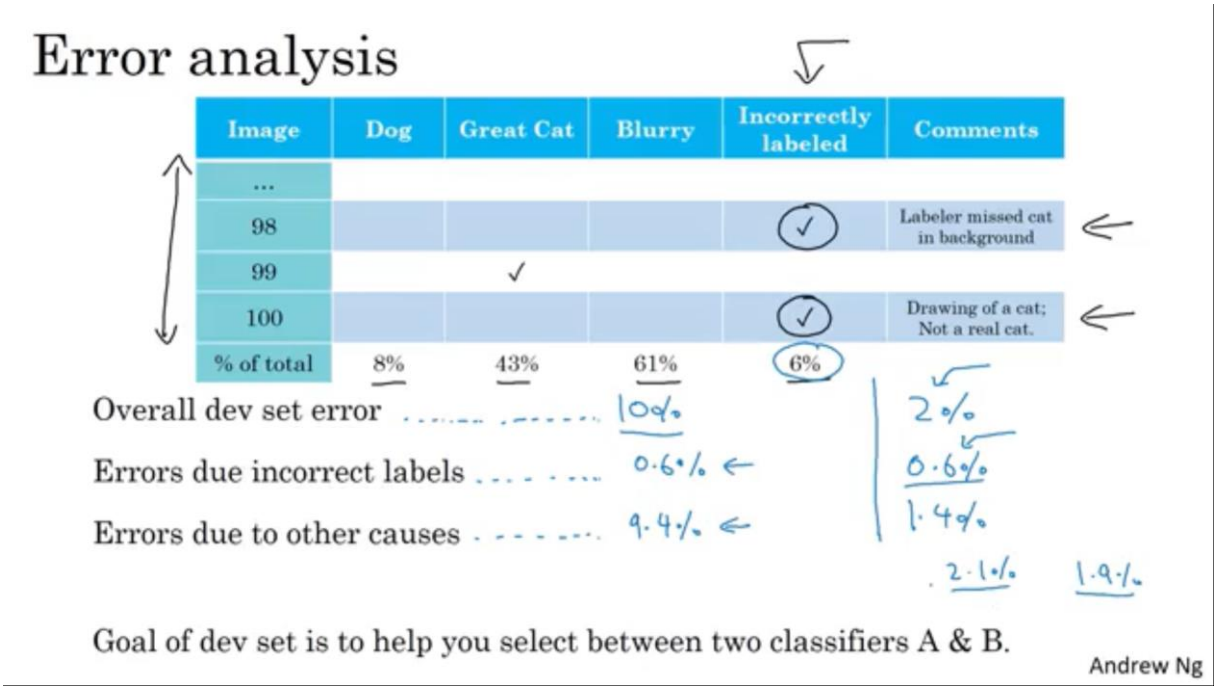
Bir cat classifier yapıyoruz diyelim aşağıda dataset içinden alınmış bazı örnekler olsun bir de bakıyoruz ki sondan bir önceki beyaz köpek cat olarak label edilmiş. Burada datayı etiketleyen kişinin hata yaptığını anlıyoruz.

Peki böyle bir durumda ne yapacağız? Öncelikle bu etiketleme hatasının training set içerisinde olduğunu varsayalım:

DL algoritmaları training set içerisinde yapılan random errors için oldukça robust yani bu büyük bir sıkıntı teşkil etmez, çünkü elimizde çok fazla data olduğu için, bu yanlış etiketlenen data doğruların içinde kaynayıp gidecektir, DL algoritmalarında genelde çok fazla data kullanıldığı için modelin bu yanlış etiketlenen dataya kendini oturtma şansı olmaz. Ancak yapılan etiketleme hatası sistematik ise yani etkiketçi bütün beyaz köpekleri kedi olarak etiketlemişse o zaman model bu yanlışı doğru kabul edip öğrenecektir.

Sonuçta training set içerisinde yanlış etiketlemeler sistematik olmadığı sürece genelde modelin performansını çok etkilemez. Peki ya bu yanlış etiketlemeler dev/test set içerisinde ise?

Eğer, dev/test içerisinde bulunan yanlış etiketlemelerin performans etkisinden endişeleniyorsak, error analysis yapıp, incorrect labeling'i bir column olarak ekleyebiliriz:



Sonuçta daha önce yaptığımız gibi dev set içerisindeki modelin hatalı ayırdettiği 100 örneğe baktık sonucunda bu hataların % kaçınının hangi sebepten kaynaklandığını bulduk.

Bu noktadan sonra hatalı etiketlemeleri düzelip düzeltmemeye burada ortaya çıkan yüzde ile karar vermek doğru olur. Örneğin yukarıda soldaki örnekte algoritmanın dev set hatası %10 olsun, ve hataların %6 sı yanlış etiketlemeden, yani totalde error'un 0.6% error incorrect labeling kaynaklı iken %9.4 diper sebeplerden, burada düzeltilebiliyorsa diğer sebeplere odaklanmak daha mantıklı görünüyor.

Diyelim ki önce diğer %9.4'lük kısıma odaklandık ve bunun büyük bölümünü çözdük sonuçta hata %2'lere kadar düştü, bu %2'nin %0.6'sı incorrect labeling den kaynaklanıyor, bu durumda dev set içerisindeki incorrect labels'ı oturup düzeltmek mantıklı olabilir.

Dev set'in amacı iki farklı model arasında seçim yapabilmek, eğer sağdaki örnekteki gibi bir error yüzdesi var ise dev set üzerinde %2.1 ve %1.9 hata veren iki model arasında seçim yapmamız çok zor olur, çünkü belki de %2.1 hatalı olan gerçekte daha iyi performans sergiliyor, incorrect labels yüzünden böyle oldu.

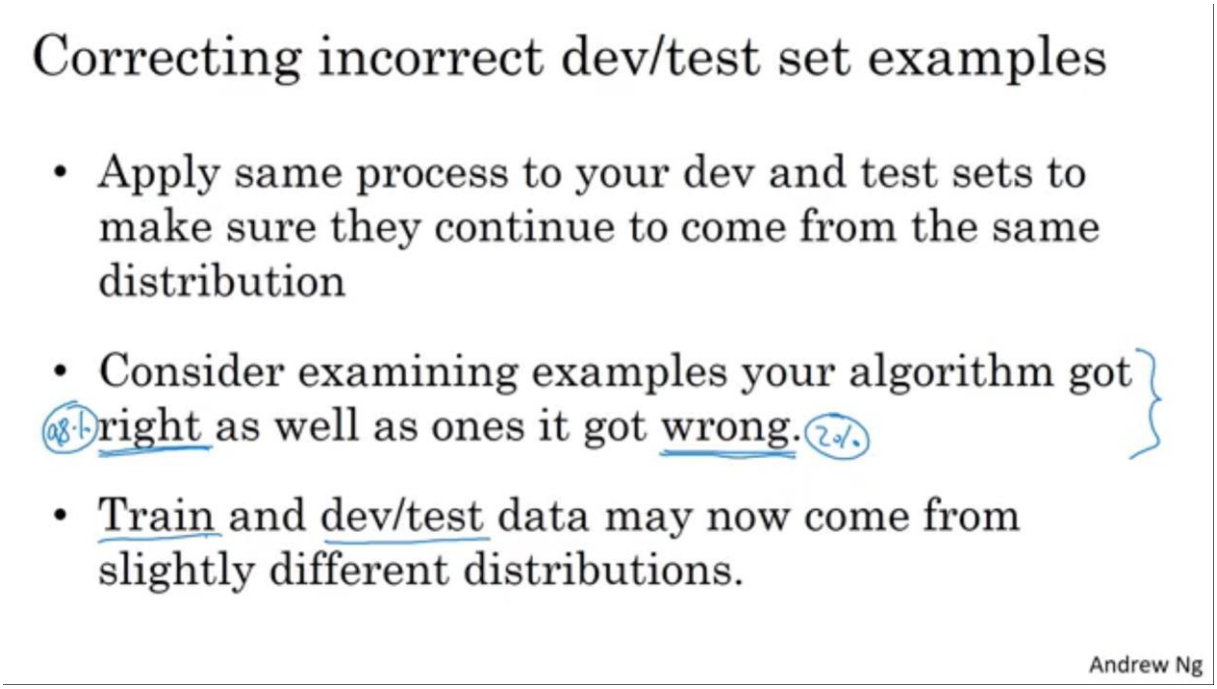
Sonuçta eğer error analysis sonucunda çıkan yüzdeler, dev set'in amacını yerine getirmesini engelleyen yüzdeler ise oturup yanlış etiketlemeleri düzeltmek mantıklı.

Diyelim ki dev set üzerindeki hatalı etiketleri düzeltmeye karar verdik bu noktada bir guideline'dan bahsedelim:

Öncelikle, eğer dev sette bir değişiklik yapacaksak aynısını test sette de yapmalıyız, daha önce de bahsetmiştik bu ikisi set aynı distribution'dan gelmeli, dev set bir nevi modelimiz için bir hedefti ve bu hedefi training set'i de kullanarak vurduğumuzda elimizdeki modelin test sete de genellenebilmesi gerek, eğer dev ile test set farklı distributionsdan gelirse, modelimizi bir nevi yanlış target için eğittik diyebiliriz.

Bu yüzden eğer dev set için label düzeltme işine giriştiysek aynısını test set için de yapmalıyız.

İkinci olarak, sadece algoritmanın yanlış yaptığı örneklere değil aynı zamanda doğru yaptıklarına da bakmalı ve yanlış etiketlemeleri düzeltmeliyiz, çünkü ikisi de hata. Ancak genelde modelin doğru bildiği örnekler yanlışlarına göre bariz daha fazla olduğu için bu adım her zaman uygulanmaz.



Son olarak, dev/test set üzerinde label correction yapıp training set'e hiç bulaşmamak mümkün, çünkü training set zaten sayıca dev/test set'e göre çok daha fazla ve learning algorithms train set'in dev ve test setten birazcık farklı bir distributiondan gelmesine karşı robustlar. Bunun için kullanılan yöntemleri bu hafta göreceğiz.

Sonuçta dev ve test set'in aynı distribution'dan gelmesi çok önemli ancak training set birazcık farklı bir distribution'dan gelebilir modelimiz bunu tolere edebilir.

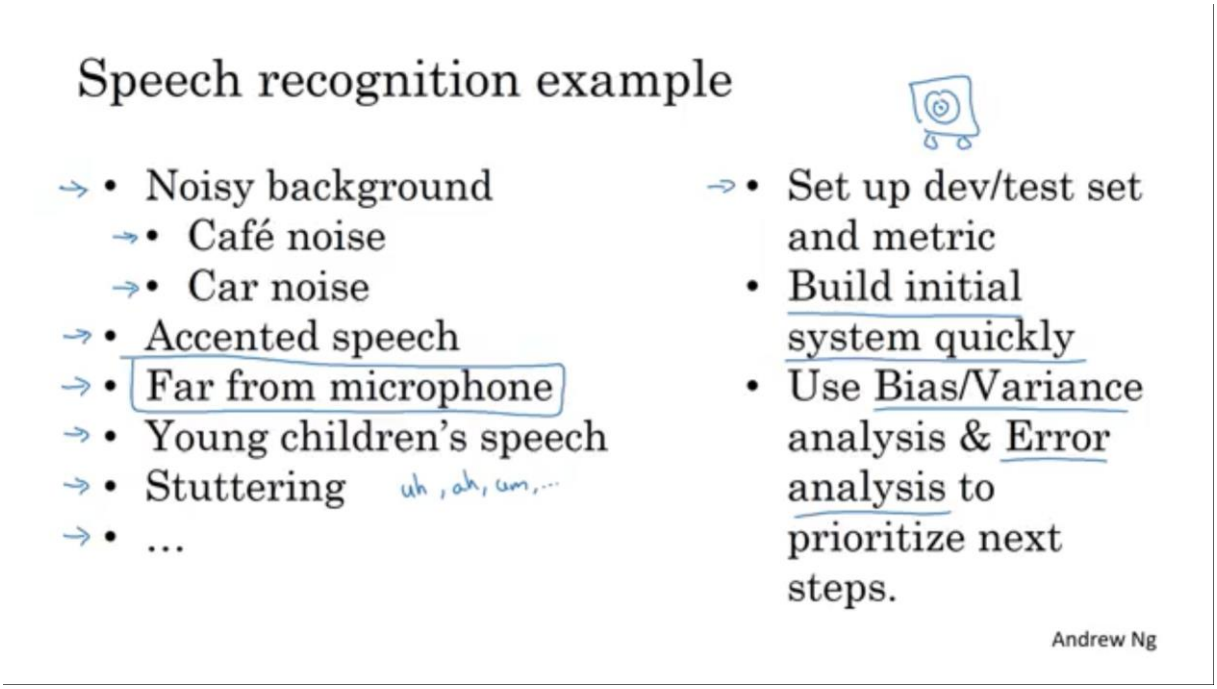
Vid 3

Build First System Quickly, Then Iterate

Eğer yeni bir problem üzerinde çalışıyorsak, öncelikle hızlıca basit bir sistem kurup daha sonra adım adım onu geliştirmeye çalışmak mantıklı olacaktır.

Diyelim ki yeni bir speech recognition sistem build edeceğiz, bu durumda öncelik verilebilecek bir sürü problem söz konusu aşağıda solda bu problemlerden bazıları görünüyor.

Genelleştirirsek yeni bir ML problemine başladığımızda birçok farklı direction söz konusu ve hepsi aslında modelimizi geliştirecek yönler, ancak hangisini seçeceğimize nasıl karar vereceğiz?



Sağ üstteki gibi bir yol izlemek mantıklı, öncelikle hızlıca bir dev/test set ve metric belirliyoruz yani modelin targetini yerleştiriyoruz, ardından yine hızlıca basit bir model eğitiyoruz, bu noktadan sonda bias/variance analizi ve error analizi ile modeli adım adım hangi yönde geliştireceğimize karar verip o şekilde ilerliyoruz.

Vid 4

Training and Testing on Different Distributions

Deep Learning modellerinin dataya karşı bir açlığı söz konusu bu yüzden giderek daha fazla takım, dev/test set ile aynı distribution'dan olsun olmasın birsürü datayı DL modeli training data olarak besliyorlar.

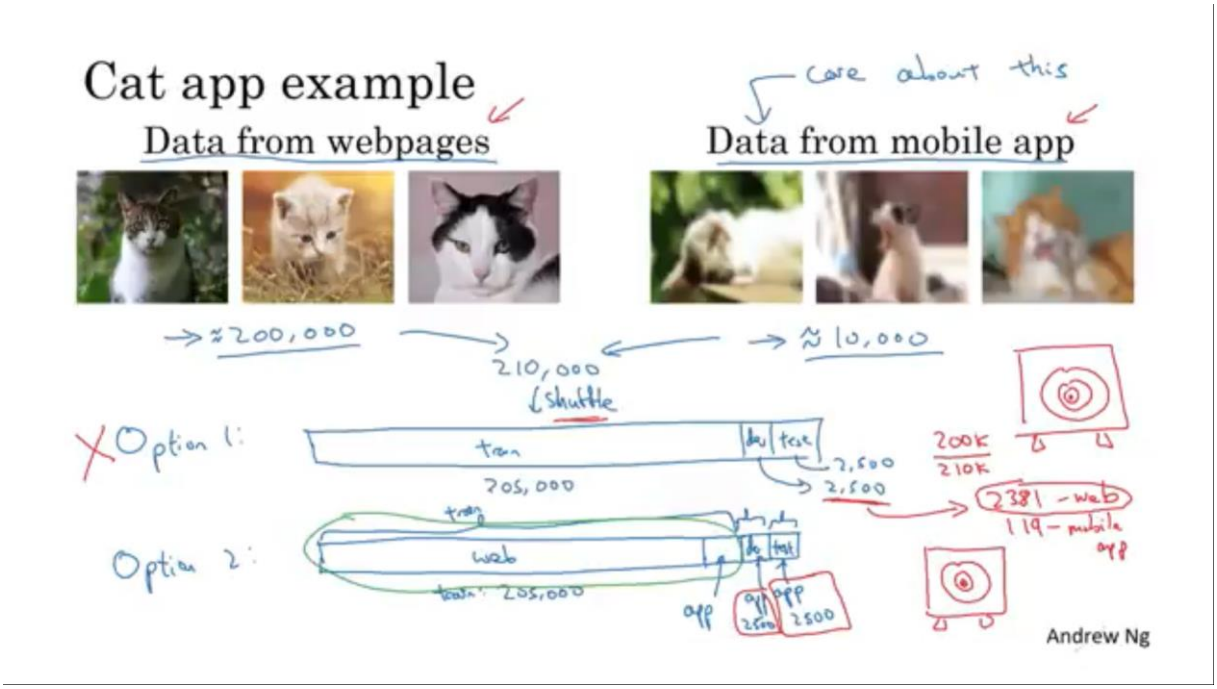
Train ve Dev/Test distributionları birbirinden farklı olduğunda yapılacak bazı şeyler var bunlara bakalım.

Diyelim ki bir mobile app yapıyoruz ve kullanıcıların yüklediği fotoğrafların cat olup olmadığını anlamaya çalışıyoruz.

Ancak henüz çok fazla kullanıcımız yok, kullanıcılardan 10 000 civarı fotoğraf örneği almışız o kadar. Internette ise birsürü fotoğraf var, 200 000 tane indirdik diyelim.

Sonuçta amacımız şu: bizim eğitilen final modelimiz, kullanıcıların yüklediği mobile app images üzerinde başarılı çalışsın.

Burada bir dilemma söz konusu, önemsediğimiz ve modelin iyi çalışmasını istediğimiz dataset sağdaki ama bundan da elimizde yeterli örnek yok, aynı distributiondan olmayan kedi fotoğraflarından ise yüzbinlerce bulabiliyoruz. Bu durumda ne yapmalıyız?



- Birinci opsiyon yukarıdaki gibi iki dataset'i birleştirip shuffle etmek daha sonra dev/test set'i ayırmak.
    - Bu opsiyonun bazı avantaj ve dezavantajları söz konusu.
    - Avantajı şu, şuanda training dev ve test setin hepsi aynı distributiondan geliyor.
    - Dezavantajı ise şu, şuanda dev/test set'in büyük çoğunluğu aslında bizim önemsediğimiz distribution'dan gelmiyor. Yani bir modelin böyle bir

dataset üzerinde iyi çalışıyor olması, mobile app dataset üzerinde de iyi çalışacağı anlamına gelmiyor.

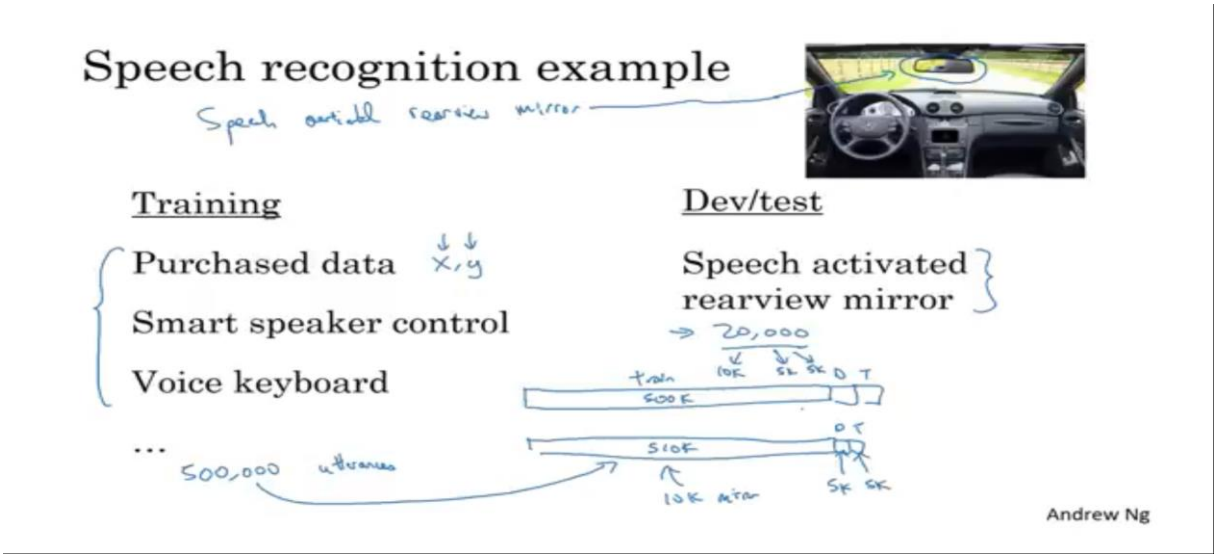
- İkinci opsiyon ise internetten elde edilen 200k datanın tamamını + mobile app datasının bir kısmını (5k) training set olarak kullanmak ve kalan 5 k mobile app datasını dev/test'e bölüştürmek.
    - Böylece target'i doğru yerleştirmiş olacağız, modelin hangi dev/test set disitribution üzerinde başarılı olması gerektiğini göstermiş olacağız.
    - Elbette bu sefer de training set ile dev/test set arasında bir distribution farkı doğacak, yani amacımızı doğru yerleştirmiş olsak da training'i biraz farklı bir data'da yapıyoruz. Yine de bu opsiyon daha iyi performans sağlar.
    - Bunu şöyle düşünelim, modelimiz bir asker ve bizim için askerin asıl performansı savaş şartlarındaki performansıdır. Elbette askerleri savaş şartlarında eğitebilsek güzel olurdu, ancak yine de askeri savaş şartlarına benzer şartlarda eğitsek bile asker savaş şartlarında iyi performans sergileyebiliyor.

Önemli olan askerin performansını savaş şartlarında (dev/test set) değerlendirebiliyor olmak, böylece askerin iyi veya kötü olduğunu söyleyebiliriz.

Ayrıca daha sonra train ve dev/test set farklı distributionsdan gelirse nasıl teknikler kullanabiliriz bunları göreceğiz.

Bir örneğe daha bakalım. Diyelim ki speech recognition ile çalışan bir rear mirror yapıyoruz. Yani kullanıcı hey mirror, show me the closest gas station diyor, mirror da gösteriyor.

Bu noktada elbette speech recognition datasına ihtiyacımız var, diğer speech recognition uygulamaları için satılan datalardan aldık diyelim, 500 000 tane data yani 500 000 ses ve karşılığında label olarak text. Bunun yanında doğrudan bizim uygulamamızla ilgili olan 20 000 data bulabiliyoruz.



Sonuçta soldaki purchased data ile sağdaki datanın distribution'u birbirinden farklı olacaktır.

Geçen örneğe benzer şekilde, soldaki datayı training set olarak kullanmak ve bizim asıl önemsediğimiz datayı ise dev/test set olarak kullanmak mantıklı olacaktır.

Buna bir alternatif olarak, 20 000 örneğin tamamını dev/test set'e koymaya gerek görmezsek, belki bunun yarısını alıp training sete ekleyebiliriz.

Vid 5

Bias and Variance with Mismatched Data

Bias ve Variance analizi ile algoritmamızı geliştirmek için sıradaki adımda ne yapmamız gerektiğine karar verebiliyoruz.

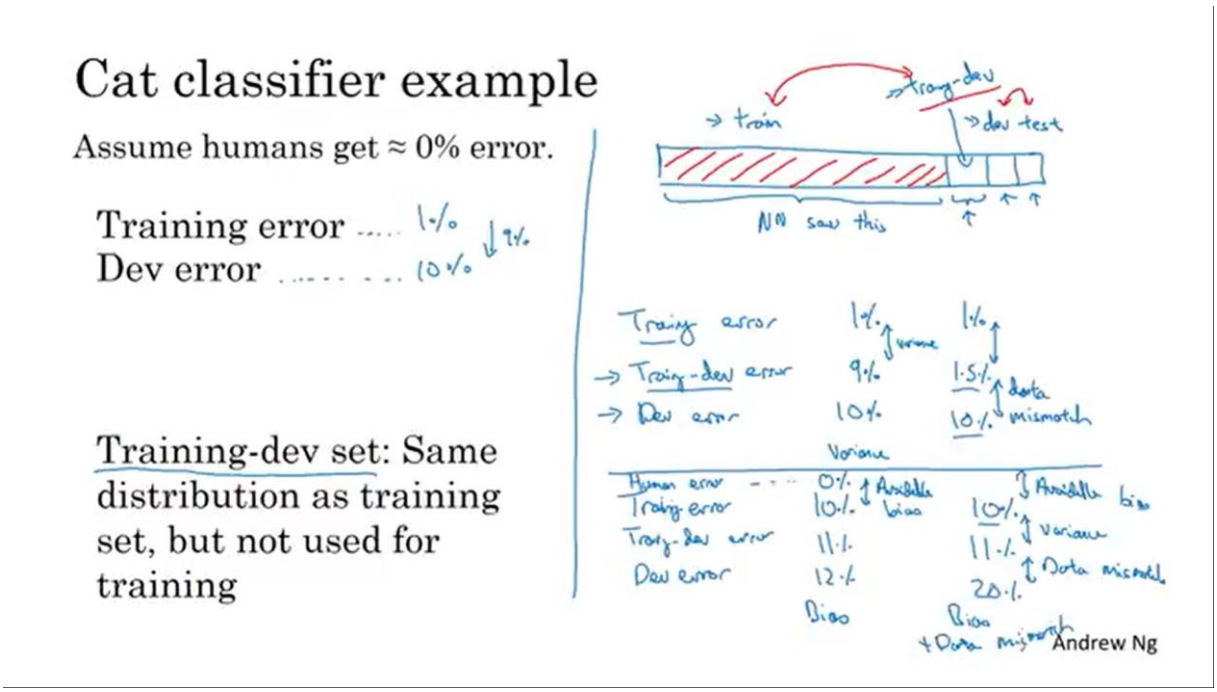
Ancak training set ile dev/test set farklı dağılımlardan gelince bias & variance analizi değişir. Bunun detayına bakalım.

Cat classifier örneğini kullanmaya devam edelim, diyelim ki insanlar %0'a yakın hata ile classification'ı gerçekleştirebiliyor, yani bu problem için bayes optimal error is nearly around zero percent.

Diyelim ki training error %1 ve dev error %10, eğer training ve dev setleri aynı distribution'dan geliyorsa bu durumda aradaki %9'luk farkın sebebini variance probleme yorardık. Ancak eğer training ve dev set farklı distributionsdan geliyorsa, artık bu çıkarımı kolayca yapamayız.

Yani belki de training set'i classify etmek daha kolaydı çünkü high resolution pictures var, ve dev set'i ayırmak zaten zor yani %10 dev set için iyi bir performans.

Böyle bir durumda bu analizin problemi şu: dev error'un train error'dan büyük olmasının iki sebebi var, birincisi modelin training set datasını görmüş ancak dev set datasını görmemiş olması, ikincisi ise dev set'in dağılımının training setten farklı olması. Hatanın yükselmesine bir yerine iki faktör etki ettiği için, artık trainerror ile deverror farkının yüzde kaçının variance'ı gösterdiğini yüzde kaçının ise sadece distribution farkından kaynaklandığını bilemeyiz.



Bu iki faktörün train error ile dev error arasındaki fark üzerinde ne kadar etkili olduğunu daha iyi anlayabilmek için bir yöntem uygularız. Yeni bir set tanımlarız, adına training-dev set diyelim. Bu set training-set ile aynı distribution'dan gelir ancak training için kullanılmaz.

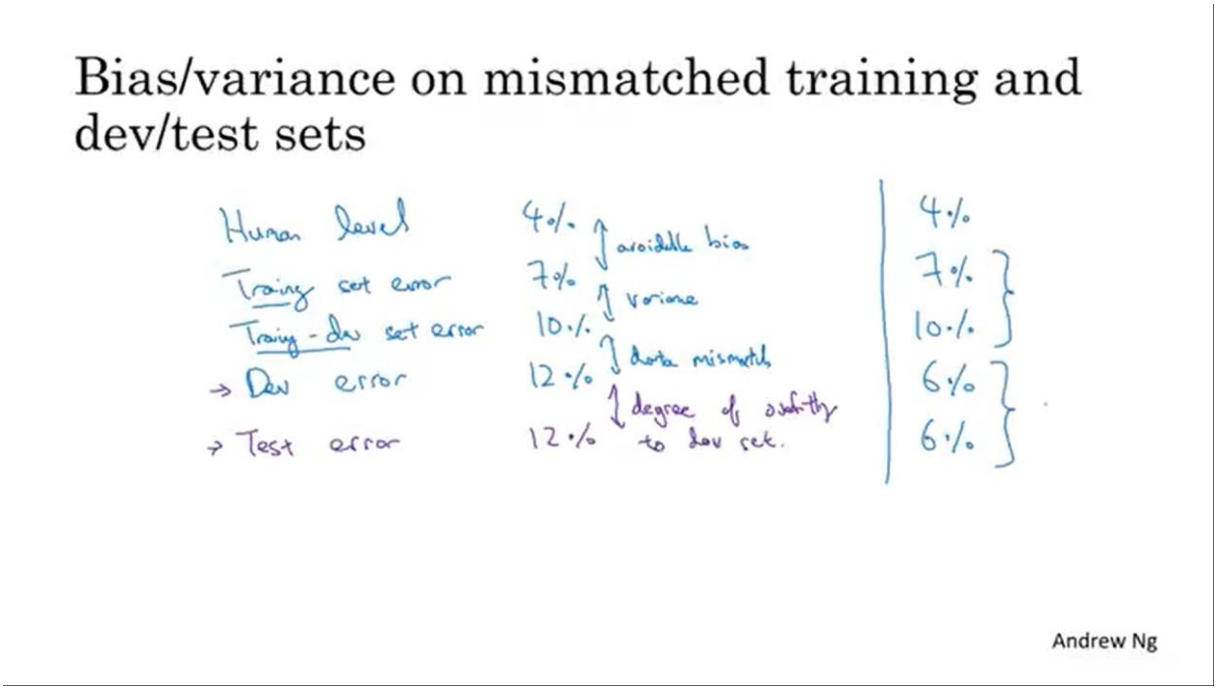
Yani sonuçta 1.train, 2.train-dev, 3.dev, 4.test set olarak 4 setimizi olur ilk ikisi ve son ikisi aynı distributiondan gelirler.

Diyelim ki böyle yaptık ve train error %1 çıktı train-dev error %9 çıktı, dev error %10 çıktı. Bu bize şunu gösterir, train error ile dev error'un arasındaki %9 farkın büyük çoğunluğu variance sebebiyle var. Nasıl anladık çünkü, train-dev error hatası da %9. Train ile train-dev arasındaki tek fark modelin train-dev'i görmemiş olması, ikisi de aynı distribution'dan geliyor.

Buna karşın, eğer train error %1, train-dev error %1.5, dev error %10 geliyorsa bu da bize asıl problemin variance değil, distribution farkı olduğunu gösterir. Buna data-mismatch problem da diyebiliriz.

Yukarıdaki fotoğrafta sağ altta görülen iki örneğe de bakarsak ilkinin bir high bias problem olduğunu görürüz. İkincisinde ise hem high bias hem de data mistmatch problem söz konusu.

Yukarıda açıklananlardan genel prensipleri anlayalım. Bakacağımız şeyler aşağıdaki veriler:



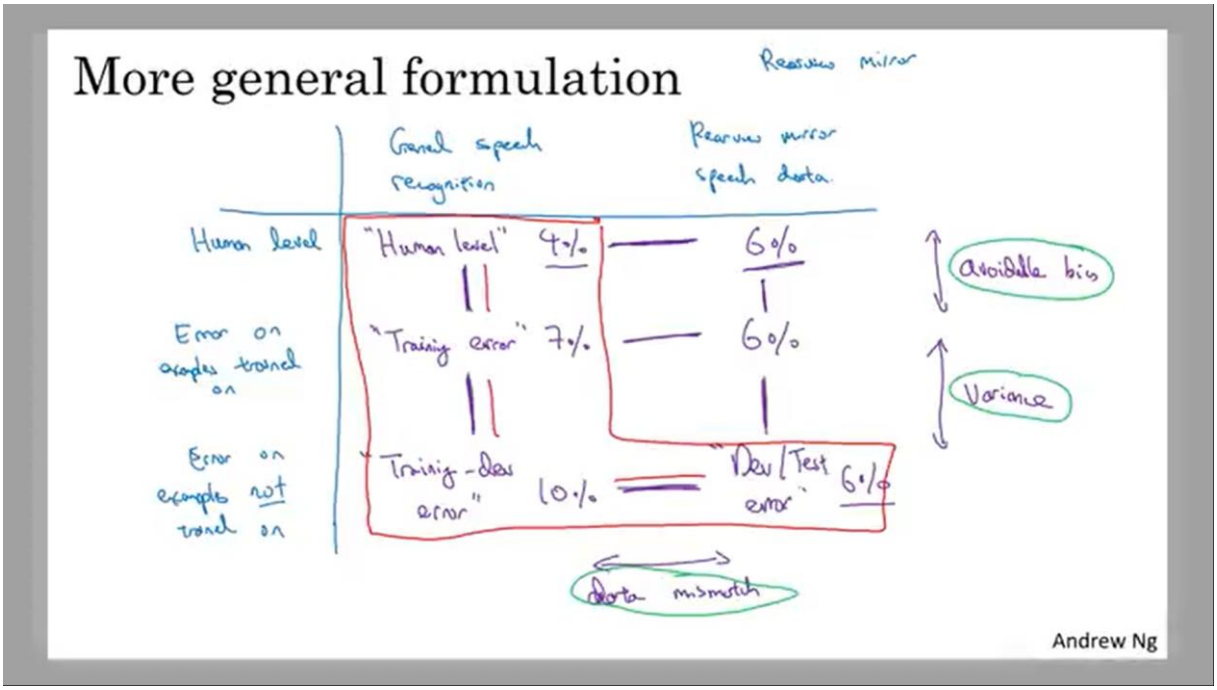
HLP, TSE, TDSE, DE ve TE.

Sonuçta bu hatalar arasındaki farklara bakarak, bias variance ve data mismatch hakkında fikir elde ederiz ve modeli nasıl geliştireceğimize karar veririz.

- HLP ile TSE farkı avoidable bias hakkında fikir verir.
- TSE ile TDSE farkı variance hakkında bilgi verir.
- TDSE ile DE farkı data mismatch hakkında bilgi verir.
- Son olarak DE ile TE farkı dev set'e yapılan overfitting derecesi hakkında bilgi verir. Sonuçta dev ve test set aynı distributiondan geliyor eğer DE ile TE farklı ise bunun sebebi overtuning'dir dev set'in boyutunu büyütmek işe yarayabilir.

Bazen yukarıdaki sağdaki gibi hata oranlarıyla da karşılaşabilinir, burada modelin training ve train-dev set üzerinde %7 ve %10 performans verdiğini görüyoruz, ancak garip bir şekilde dev ve test set üzerinde daha iyi performans veriyor, bu bazen olabilir, demekki training set dev set'den daha zormuş, model training sette %10 performansa ulaşınca dev-set üzerinde gayet iyi performans gösteriyor.

Bunu daha iyi anlamak için rear view window örneğine tekrar dönelim. Aşağıdaki gibi bir tablo ile, bias variance ve data mismatch arasında bazı insights elde etmek mümkün.



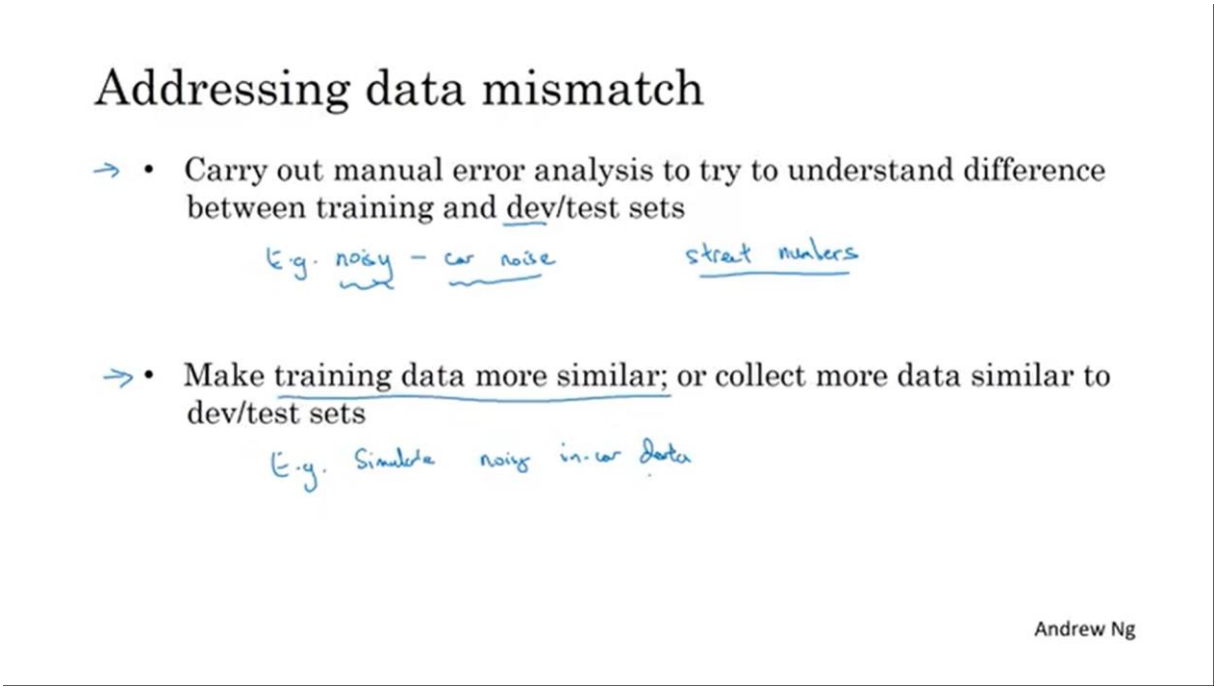
Bu noktada önemli bir soru şu: Bias veya Variance ile karşılaştığımızda nasıl bir çözüm üreteceğimizi biliyoruz peki ya data mismatch problemine nasıl bir çözüm üretiriz? Bunu diğer başlıkta göreceğiz.

Vid 6

Addressing Data Mismatch

Diyelim ki training set ile dev/test set farklı distributionsdan geliyor ve analizimiz sonucunda (train-dev set kullanarak) ortada bir data mismatch probleminin olduğuna kanaat getirdik. Bu durumda ne yapmalıyız?

Bu durumda tam olarak sistematik bir çözüm yok ama, deneyebileceğimiz bazı şeylere bakalım:

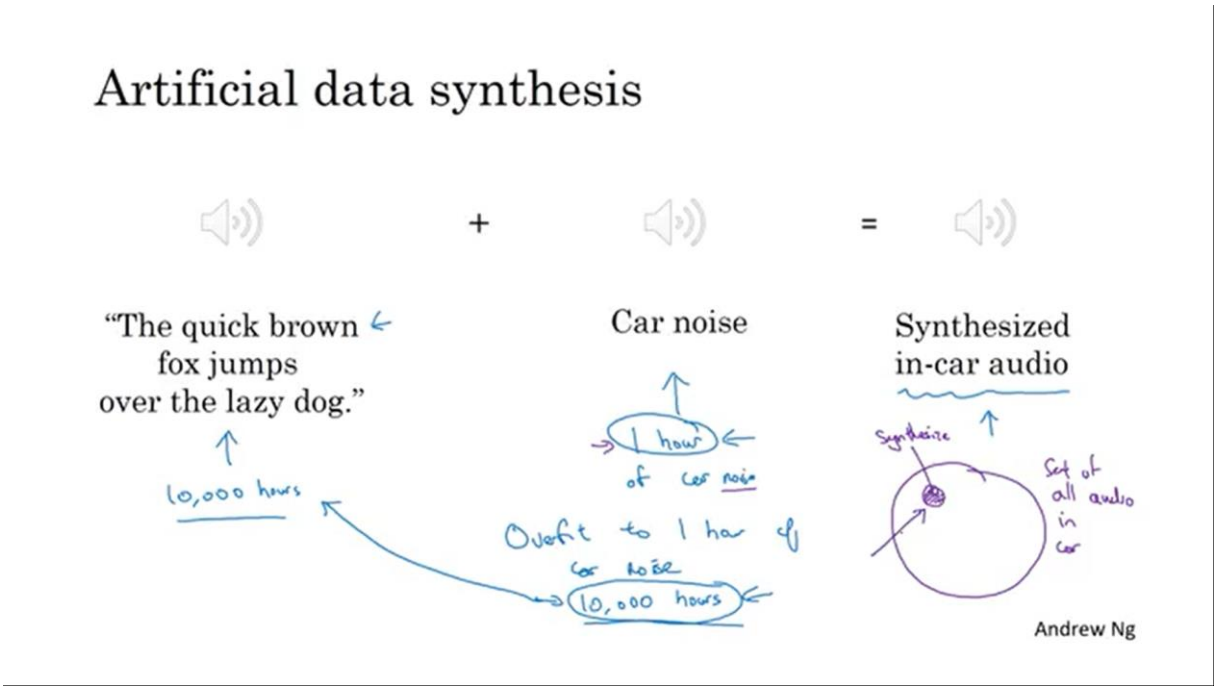


Yapabileğimiz ilk şey manual error analizi ile training ve dev/test set arasındaki farkı anlamaya çalışmak.

Bu setlerin farklı distributionsdan gelmesine neden olan etken nedir? Yani rear mirror örneği için, training set ve dev set içindeki datalara bakarız ve bu dataların farkı ne onu anlamaya çalışırız. Mesela dev set örneklerinde car noise söz konusu olabilir, veya street numbers seslerini ayırmak model için zor geliyor olabilir.

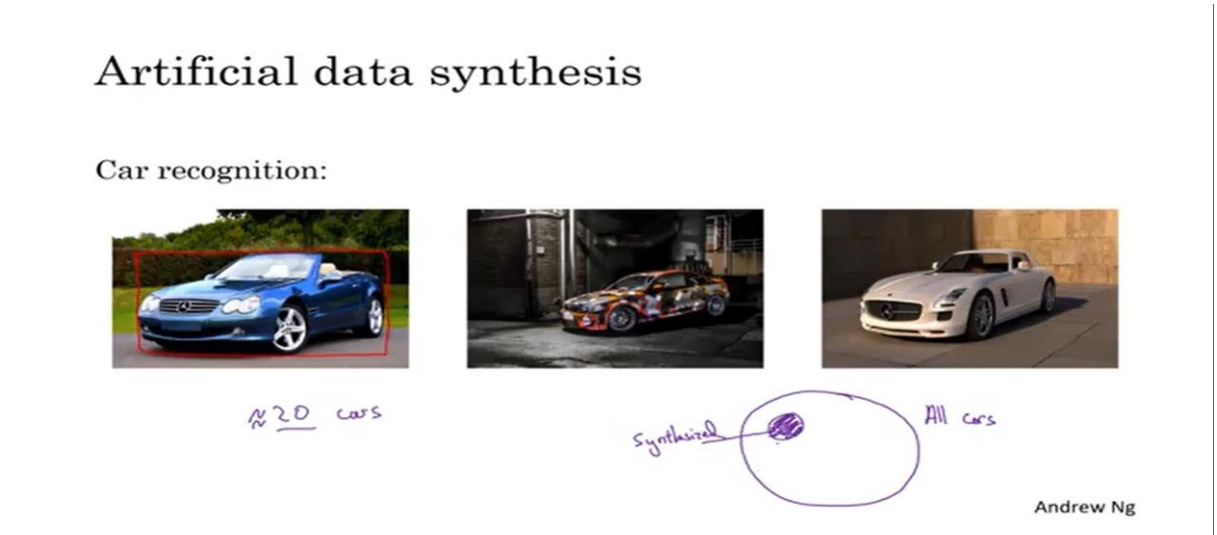
Bu farkı kavradıktan sonra amacımız training data'yı dev/test sete daha benzer bir hale getirmek veya training set için dev/test set'e benzeyen daha fazla örnek bulmak. Yani belki dataya car noise arkaplanı eklemeye çalışabiliriz veya street numbers etiketli data bulmaya çalışabiliriz.

Eğer amacımız training data'yı dev data'ya benzer hale getirmekse bunu nasıl yapabiliriz? Kullanılabilecek tekniklerden biri Artificial Data Synthesis'dir. Temelde yapılan şey, dev set ile farklı distributiondan olan training set datasını alıp, ona car noise ekleyerek dev set'in distribution'ına yakınlaştırmaktır.



Burada önemli bir nokta şu, diyelim ki artificial data sentezi yapıyoruz, ancak bunu yaparken 10 000 saatlik training set örneğimizi transform etmemiz için sadece 1 saatlik unique car noise örneği kullanıyoruz, eğer 1 saatlik veriyor tekrarlayıp tekrarlayıp 10 000 saatlik veriye eklersek, insan kulağına problem yok gibi gelse de algoritma 1 saatlik car noise'a overfit edebilir, böyle olunca da diğer car noise için model iyi çalışmaz hale gelir. Yani burda yaptığımız şey araba içindeki seslerin sadece küçük bir subsetini alıp datayı ona göre sentezlemek oldu. Bunun yerine mümkünse 10 000 saatlik car noise bulup kullanmak daha iyi sonuç verecektir.

Bir başka benzer örnek car recognition problemi için verilebilir, amacımız en sol alttaki gibi car detect edip etrafına bir window oturtmak olabilir, bu amaçla car examples kullanmak istiyor olabiliriz, bu yüzden gidip CGI car fotoğraflarını kullanmak isteyebiliriz, belki bir oyundan car fotoğrafları alıp kullanıyoruz, burada da insan gözüne bu okay görünebilir ama modelimiz computer generated car images'e overfit edebilir, belki oyunda 20 çeşit car var ve bize bu yeterli görünüyor ama gerçek hayat 20 car'dan çok daha fazlasını içeriyor.



Vid 7

Transfer Learning

Deep learningteki güçlü fikirlerden biri Learning Transfer'dir. Bu temelde bir task için eğitilen NN'in edindiği bilgileri başka bir task için kullanma fikridir. Yani mesela object detection için eğitilen bir NN'ümüz olsun, bu network'ün edindiği bilgilerin bir kısmını (yani network'ün bir kısmını) alıp bir başka iş için mesela reading x-ray scans için kullanabiliriz.

Diyelim ki image recognition için aşağıdaki gibi bir NN eğittik, x image'ımız y ise hangi object olduğu (bird, cat, etc.) Diyelim ki bu task için öğrenilen bilgilerin bir kısmının radiology diagnosis task'i için yardımcı olabileceğini düşünüyoruz bu durumda son layer'ı ve weight'ini silip yerine randomly initialized bir weight ve bir layer ekleyebiliriz, daha sonra NN'i radiology dataset için tekrar eğitiriz artık bu NN'i radiology diagnosis için kullanırsak, önceki layerlarda image recognition'dan elde edilen bilgiler işlevsel olacaktır.

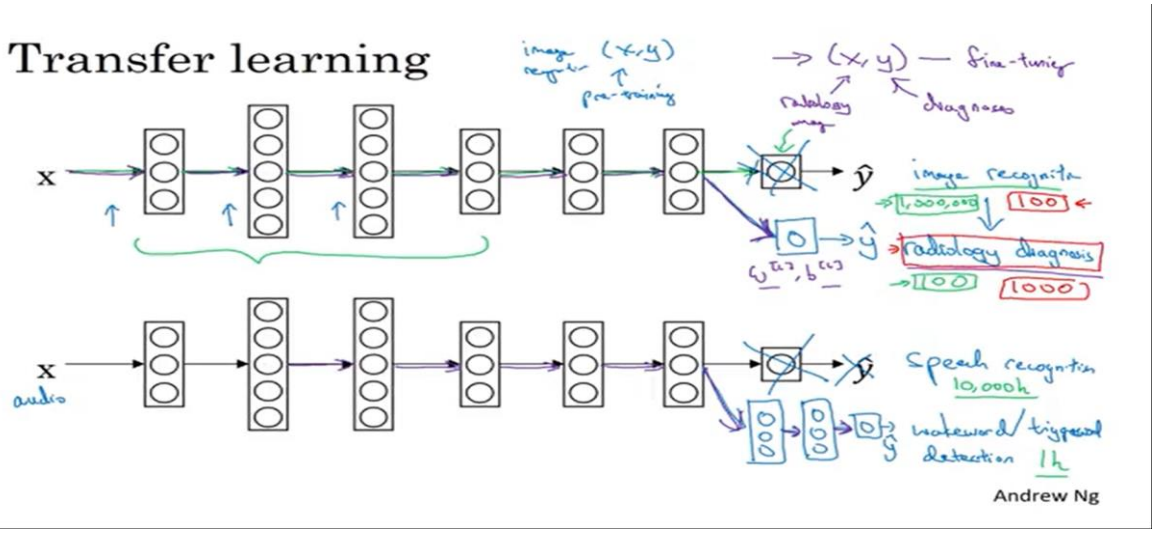
Eğer yeterince radiology diagnosis datamız yoksa sadece son layer'ın weight'ini eğitiriz, diğer weightler fix kalır, ya da belki son birkaç layer için eğitiriz. Eğer yeterli datamız varsa tüm layerları da eğitebiliriz.

Eğer tüm layerları tekrar eğitirsek, NN'in image recognition için eğitilmesine pre-training diyebiliriz, daha sonra radiology data için tekrar eğittiğimizde buna da fine-tuning deriz.

Temelde yapılan şey image recognition için eğitilen NN'in edindiği bilgileri radiology diagnosis problemine transfer etmek.

Altta yakan inution ise şu, image recognition için büyük bir datasetimiz varsa modeli önce bununla eğittiğimiz zaman ilk layerlar edge dedection, curve detection, vb gibi feature bulma konusunda ön eğitim almış oluyorlar, sonuçta radiology probleminde de öncelikle bu feature'ların çıkarılması gerekiyor. Dolayısıyla modelimiz en azından bu feature'ları nasıl çıkaracağını kavramış oluyor daha sonra bu eğitiminin üstüne bir de radiology eğitimi alıyor.

Şöyle düşün model önce temel fotoğraf özellikleri eğitimi alıyor, sonra üzerine master olarak radiology fotoğraf özellikleri eğitimini alınca, alanında uzman oluyor. Temeli olmadan radiology diagnosis eğitimi de istenilen performansı sağlamayabilir.



Yukarıdaki diğer örnek ise speech recognition system için verilmiş. Diyelim speech recognition için bir model eğittik, input auodio, output ise karşılık gelen transcript. Şimdi ise wakeword/triggerword detection system geliştirmek istiyoruz, yani Hey Siri dediğinde uyansın gibi.

Bunun için önceki örneğe benzer şekilde pre-trained network'ün outputunu keseriz, yerine bir veya daha fazla layer ekleriz sonra yeni örneklerin sayısına bağlı olarak sadece yeni layerları veya tüm layerları train ederiz ve learning transfer uygulamış oluruz.

When does transfer learning make sense? Yani bunu ne zaman kullanmalıyız?

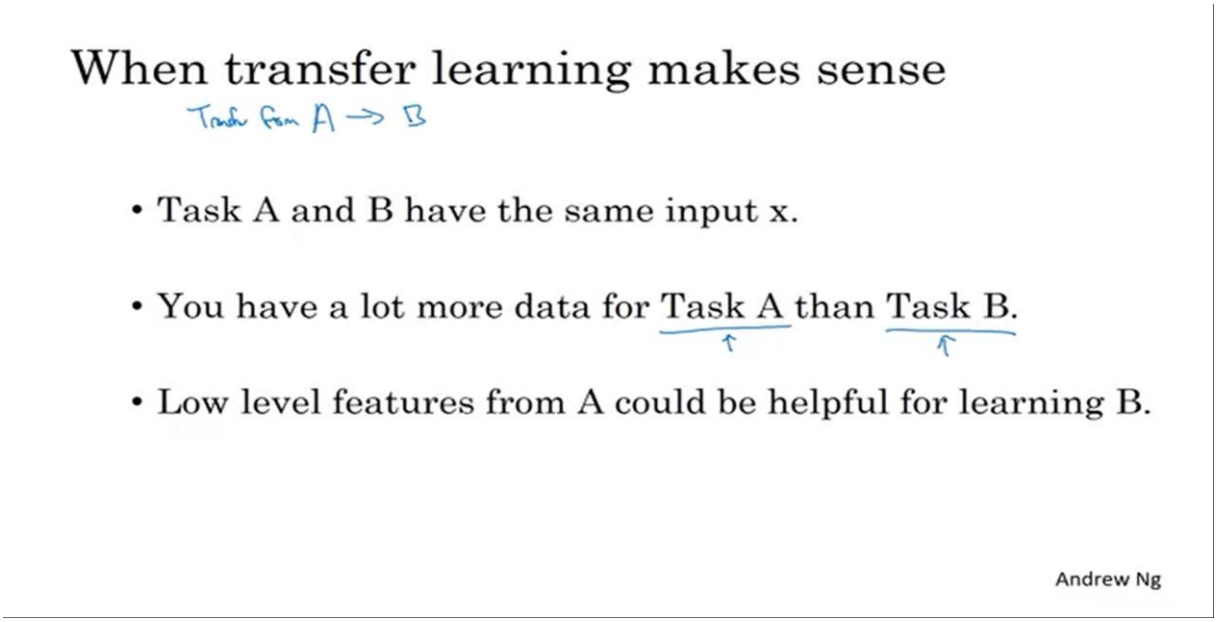
Transfer learning'in mantıklı olduğu zaman, when we have a lot of data for the problem we are transferring from and relatively less data for the problem we are transferring to.

Yani örneğin image recognition için 1m datamız var, bu kadar data ile bir image'ın low level featurelarını yakalamayı model gayet iyi öğrenebilir, radiology problem için belki elimizde sadece 100 data var. Bu durumda modelin image recognition için 1m datadan öğrendiği bilgileri radiology probleme aktarmak mantıklı olacaktır.

Benzer şekilde speech recognition için belki 10 000 hour data ile model human sound'un temel feature'larını kavradı ve trigger word için 1 hour data var tek başına bir model eğitmek için yeterli olmayabilir, speech recognition için eğitilen modelin bilgilerini transfer etmek mantıklı olacaktır.

Fakat bu durumun tersi geçerliyse mesela image recognition için 100 örnek varsa ve radiology diagnosis için de 100 veya 1000 örnek varsa bu durumda transfer learning'in bir zararı olmaz ama bir işe de yaramaz, bunu şöyle düşün sonuçta amacımız radiology diagnosis için başarılı bir model eğitmek bunun için radiology datanın bir örneği image recognition'ın bir örneğinden (cat, dog, bird etc. pictures) çok daha değerli, ancak ortada milyonlanlarda image recognition data olunca, her ne kadar bir tanesi önemsiz olsa da totali daha önemli hale geliyor.

To summarize, when does TL makes sense?



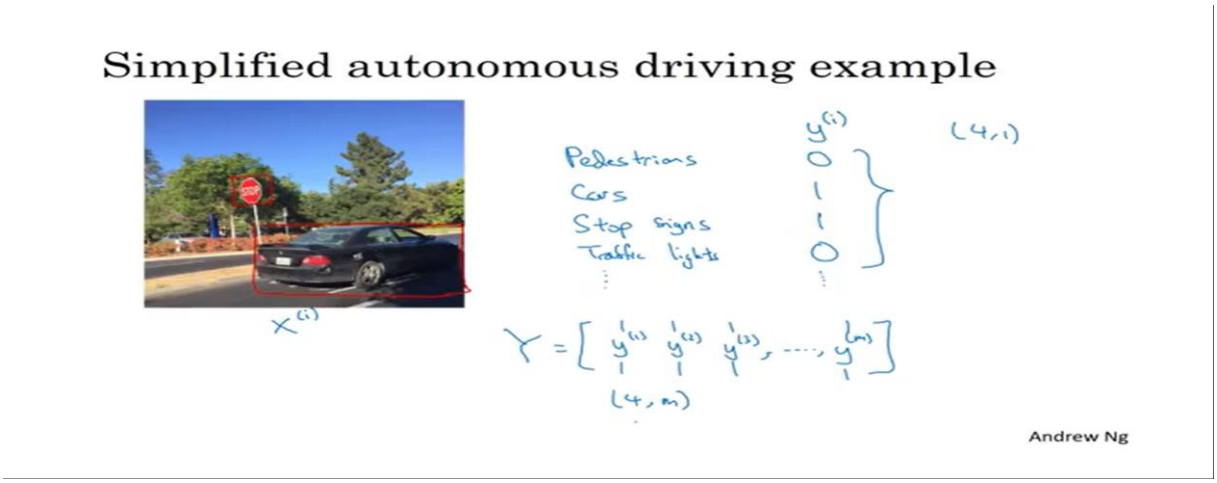
Eğer Task A'den Task B'yi öğreniyorsak yukarıdaki durumlarda transfer learning uygulamak mantıklı.

Vid 8

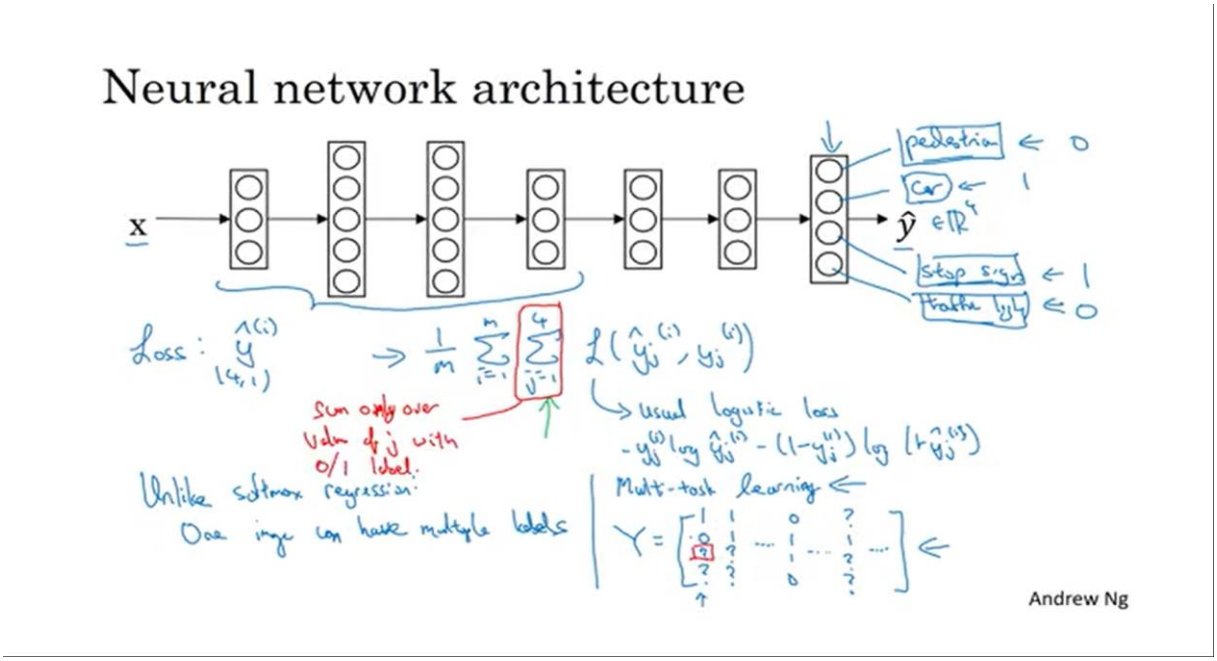
Multitask Learning

Transfer learning ile task A'dan öğrenilen bilgiyi task B için kullanıyorduk. Multitask learning için ise simultaneous olarak bir NN'i birden fazla task için eğitiyoruz, ve hopefully each of these tasks help all of the other tasks.

Bir örneğe bakalım. Diyelim ki amacımız autonomous driving için aynı yayaları, araçları, trafik işaretlerini ve ışıklarını tanıyan bir sistem geliştirmek. NN'e alttaki gibi bir input image girdiğinde sağ alttaki gibi bir output çıkacak, tek bir label yerine 4 label olacak. 0 1 1 0. Daha önceden böyle bir uygulama yapmadık, bunun yerine 4 label olsa bile her bir y için yalnızca biri 1 olabiliyordu yani bir başka deyişle. Tek bir fotoğraf ile yalnızca tek bir obje tanıyorduk.



Bu amaç için aşağıdaki gibi bir NN eğittiğimizi düşünelim. Output layerdaki her bir node başka bir objeyi predict etmeye çalışıyor.



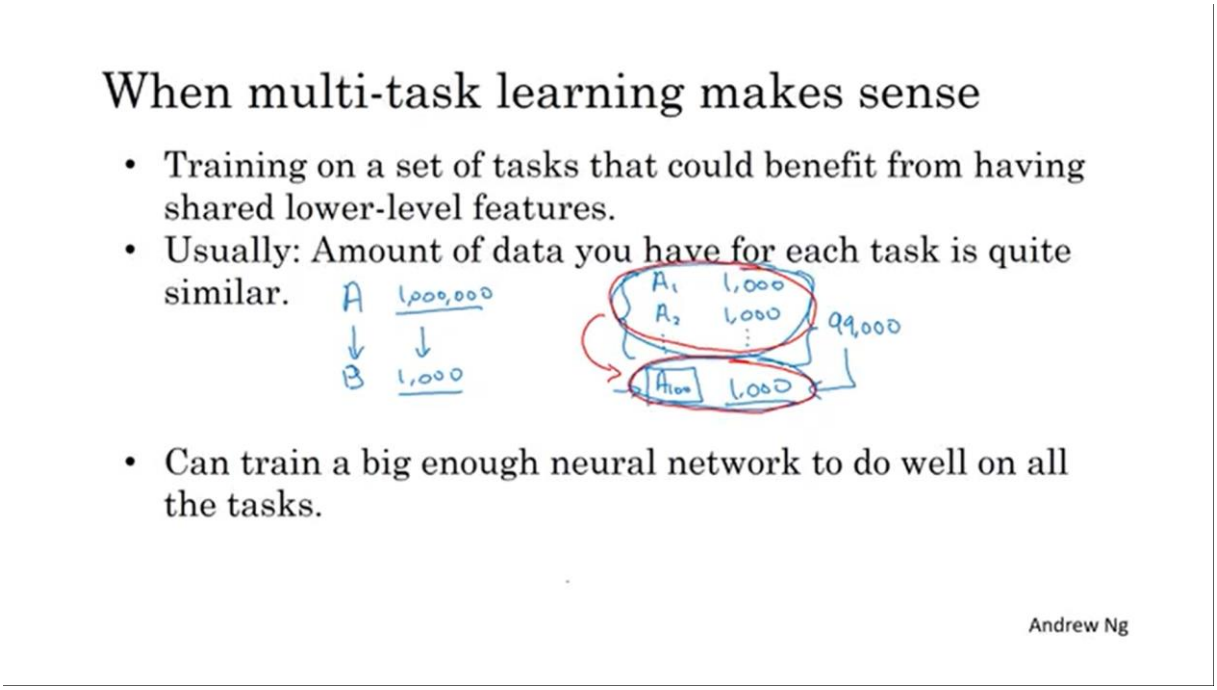
Loss function da basitçe her output unit için logistic loss'un toplamı şeklinde tanımlanabilir.

Softmax regression da bir image sadece tek bir label olabilirdi yani output layer da tek seferde yalnızca bir adet ON unit olabilir, burada ise multiple ON olabiliyor.

Böyle bir loss function ile eğitim yapıyorsak, yaptığımız şey Multitask Learning. 4 farklı task için aynı anda tek bir model eğitiyoruz. Buna alternatif olarak yapabileceğimiz bir şey 4 farklı model eğitmek olabilirdi burada önceki layerlarda her bir task için elde edilen low level feature dection'ın diğer tasklere yardımcı olmasını ve 4 ayrı model eğitmekten daha iyi performans vermesini umuyoruz.

Eğer bazı labellarımız yukarıda sağ alttaki gibi girilmemişse multitask learning yine de çalışır.

So when does multi-task learning makes sense? Aşağıdaki 3 şey sağlandığında multi-task kullanmak mantıklıdır.



Ben ikinci maddeden şunu anlıyorum, bunun mantıklı olması için her task için ayrı bir dataset olması lazım ve her set aşağı yukarı birbine eşit olması lazım. Yani yukarıda verilen 4 çıkışlı detection probleminde multi-task learning'in 4 farklı model eğitmekten daha başarılı olması için; pedestrian, car, traffic signs and lights çıkışlarının her biri için bağımsız dataset'im olması lazım. Tabi bu dataset'in içinde bazı örnekler hem pedestrian hem car vb içerecek. Bu durumda her bir çıkış için bir model eğitip yalnızca 1000 example'dan yararlanmak yerine 4000 example ile multi-task learning uygularız ve modelimiz diğer 3000 example'dan öğrendiklerini de prediction için kullanabilir. Ancak elimizde toplam sadece 1000 example varsa he multitask yapmışzı he 4 ayrı model eğitmişiz bence farkı yok, multi-task daha kısa sürebilir.

Multitask'in performansı sadece NN yeterince büyük değilse düşürebilir, onun dışında pek düşürmez.

Vid 9

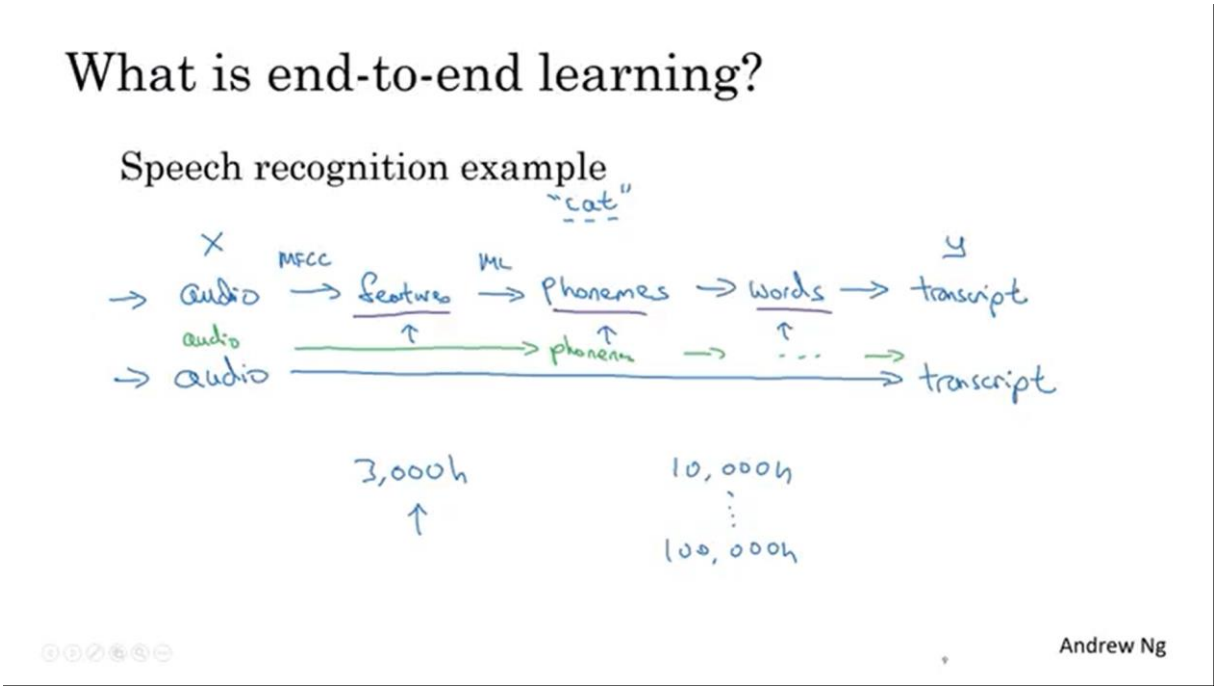
What is End-to-End Deep Learning?

Deep Learning era ile end-to-end deep learning uygulamaları giderek artıyor. Peki nedir bu end-to-end deep learning?

Basitçe açıklamak gerekirse, bazı uygulamalar birden fazla step gerektirebilir, end-to-end (ete) DL ile ise tüm stepler yerine tek bir NN ile işi hallederiz.

Bazı örneklere bakalım. Diyelim ki bir speech recognition problemi ile uğraşıyoruz, nihai amacımız input olarak alınan audio için transcript üretmek.

Böyle bir problem için traditional yaklaşım aşağıda görülen ilk yaklaşımdır önce MFCC yöntemi ile audio'dan bazı hand-designed features elde edilir, daha sonra bu low level features'a ML uygulanarak audio clip içerisindeki phonemes (yani alfabetik ses birimleri) bulunur. Daha sonra kelimeler elde edilir ve son olarak transcript elde edilir.



End to end deep learning yaklaşımı ile ise tüm bu adımları pas geçip onun yerine tek bir deep learning modeli ile verilen audio'dan doğrudan transcript elde edilebilir.

Bu yöntem deep learning ile popular oldu bundan önce kariyerinin büyük bölümünü geleneksel yaklaşımdaki tek bir adıma ayıranlar oluyordu, bu yüzden bu yeni end-to-end yaklaşımı kabüllenmekte zorlandılar.

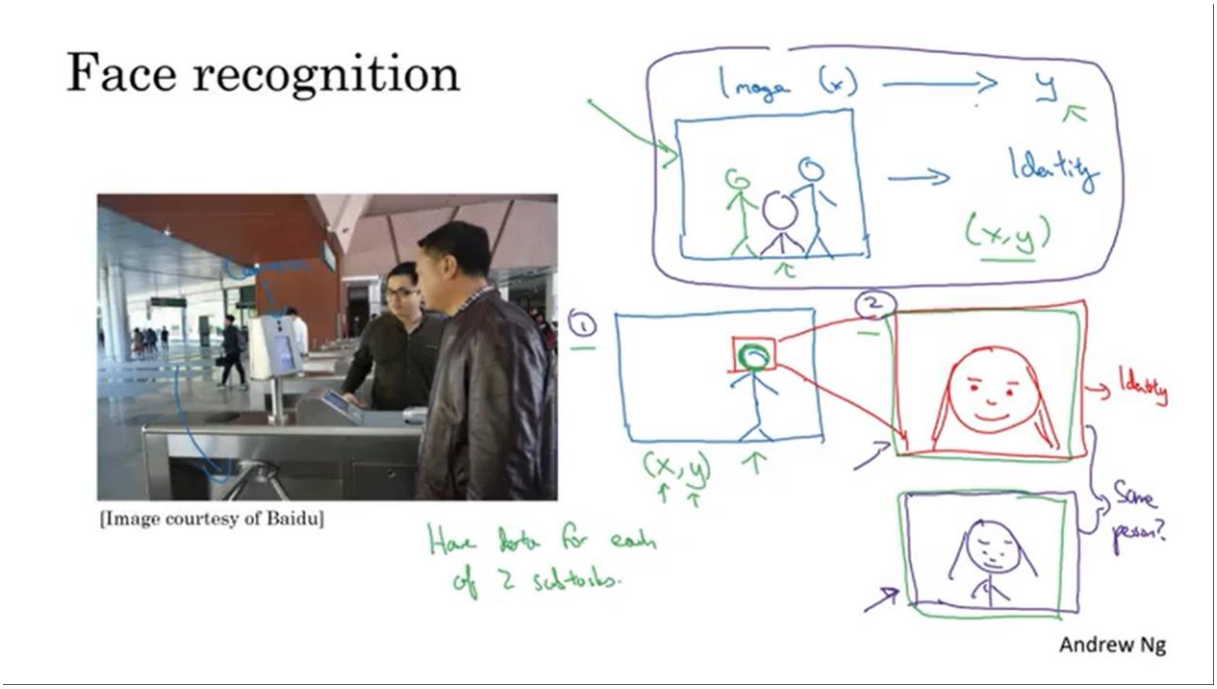
Ancak her problem için end to end deep learning uygulayamıyoruz, end to end deep learning'in çalışabilmesi için çok fazla sayıda dataya ihtiyacımız var. Yani yukarıdaki örnek için mesela 3kh civarı speech datamız varsa traditional approach iyi çalışır, ancak 10k-100k civarlarında datamız varsa, bu end-to-end bir model eğitmek için yeterli olabilir.

Ayrıca data ne az ne çoksa, traditional ile ete arası bir yaklaşım uygulanabilir, yani tüm adımları bypass geçmek yerine bazıları bypass geçilebilir.

Şimdi bir başka örneğe bakalım. Çinde gate'ler RFID yerine face recognition ile çalışıyor, turnikelere kart okutmak yerine kameralar yüz tanıyor ve geçiyorsunuz.

Peki böyle bir sistemi nasıl oluşturabiliriz?

- Birinci yaklaşım, doğrudan image'a bakıp identity'e eşleyen bir function öğrenmeye çalışabiliriz. Ancak bu iyi bir yaklaşım değil.
- Neden iyi değil? Çünkü doğrudan ham image verisine baktığımızda kişinin fotoğraf karesine nasıl girdiğiyle ilgili bir çok olasılık var, kameraya yakın olabilir, uzak olabilir, belki uzun boylu birisi, belki absürt bi kıyafet giymiş vesaire.
- Bu yüzden burda end-to-end yerine multistep approach uygulamak daha mantıklı.
- İlk adımda, bir face detection algoritması kullanılabilir, bu bildiğin gibi gayet basit bir uygulama, daha sonra ikinci adımda detect edilen face image input olarak alınır ve identity eşleyen bir NN için input olarak kullanılabilir.
- Bu ikinci adım için NN'e çekilen fotoğraf'ın yanında kayıtlı ID'lerin fotoğrafları input olarak verilir, eğer herhangi biriyle eşleşiyorsa, o id'yi döndürür.

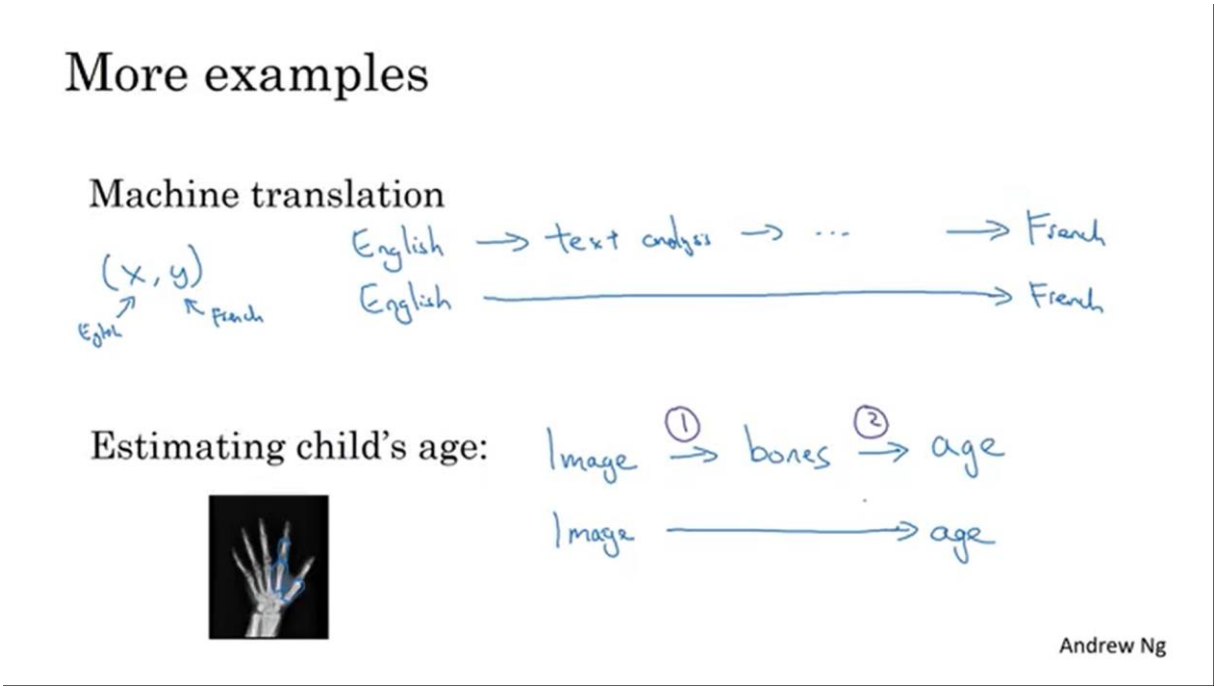


Peki neden bu durumda end-to-end yerine 2 step approach daha iyi çalışıyor? Bunun iki nedeni var:

- Problemi böldüğümüzde ortaya çıkan 2 alt problemin çözümü çok daha kolay.
- Subproblems için elimizde çok daha fazla data var, zaten bu yüzden çözümü daha kolay. Hem face detection için hem de yüzden identity eşleme için tonlarca data var ancak, rastgele boydan fotoğraflardan id tanımak için yeterli data yok.
- Bu yüzden end-to-end approach data eksikliği sebebiyle burada işe yaramaz. Ancak gelecekte eğer yeterli data toplanırsa, elbette end-to-end approach daha iyi çalışabilir.

İki farklı örneğe daha bakalım, birincisi machine translation, bu problem de geleneksel olarak complicated pipelines'a sahip, önce bir dilde cümleyi alıyor, bir çok adımdan geçtikten sonra translation yapılıyor.

Ancak elimizde tonlarca doğrudan english-french pairs sentences datası olduğu için deep learning ile end-to-end approach uygulanabiliyor.



İkinci örnek ise bir x-ray image'a bakarak çocuğun yaşının tahmin edilmesi. Bu yöntem çocuk gelişimini takip edebilmek için kullanılıyor.

Bu problem için uygulanan yöntem şöyle, önce image alınır, bones'lar ayrılır, daha sonra hangi bone'un hangi uzunlukta olduğu bilgisi çıkarılır ve bir bones-age lookup table ile çocuğun yaşı estimate edilir.

Bu yöntem şuan gayet iyi çalışıyor, buna alternative olarak end-to-end approach kullanılabilir ancak şuan elimizde yeterli x-ray image --- age pair verisi yok bu yüzden şuanda bu kullanılmıyor.

Sonuç olarak end-to-end kullanmak daha iyi ancak bunu kullanabilmek yeterli verinin olup olmamasıyla ilgili bir şey eğer yeterli veri yoksa end-to-end approach kullanamıyoruz, model o kadar karmaşık bilgiyi az veriyle öğrenemiyor.

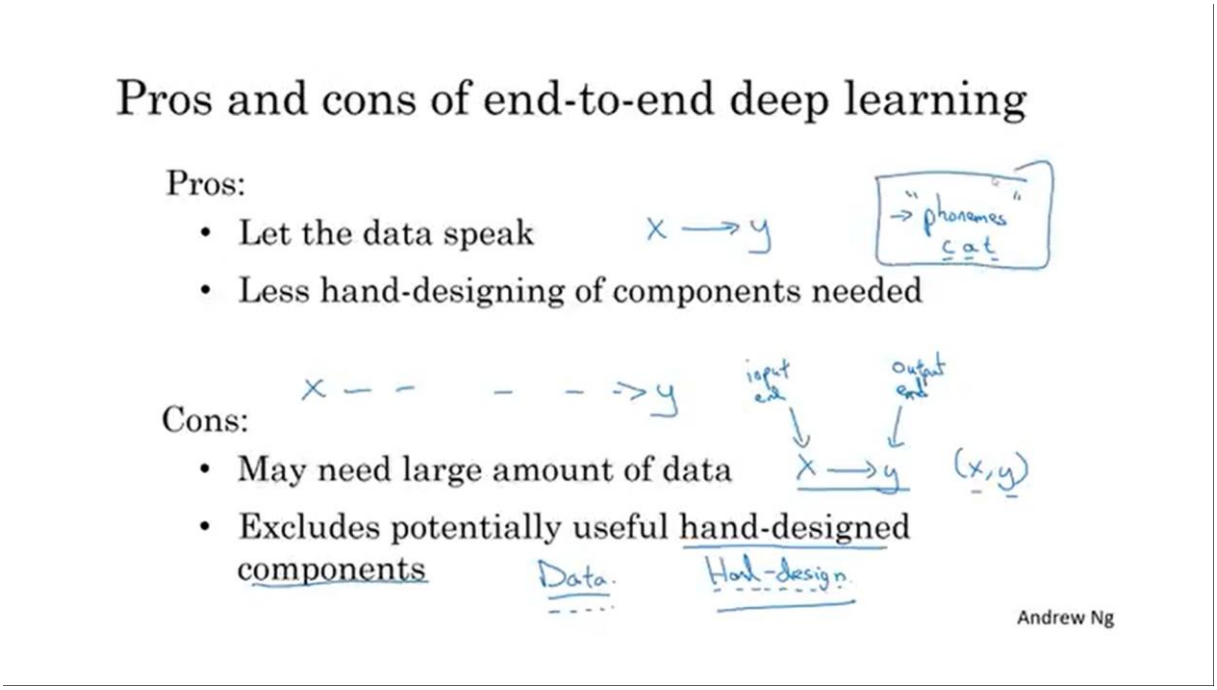
Vid 10

Whether to use End-to-End Deep Learning?

Diyelim ki bir ML sistemi oluşturuyoruz, acaba end-to-end approach kullanmalı mıyız yoksa kullanmamalı mıyız? Buna sistematik bir yaklaşım getirmeye çalışalım.

Öncelikle end-to-end deep learning'in artı ve eksilerine bakalım.

- Birinci artı şu, modelin öğrenmesini istediğimiz fonksiyon neyse model onu insan önyargısı olmadan sadece data'ya bakarık öğrenecek, belki insandan daha iyi bir yol ile öğrenecek. Yani örneğin speech recognition için, biz problem manuel olarak parçalara bölüyoruz ve aslında modelimiz için yararlı olabilecek özellikleri elle çıkarıp modelimize besliyoruz (mesela phonemes), ancak belki model ham data'yı kullansa, bizim ona verdiğimiz feature'lardan daha farklı ve kesin feature'lar elde edecek ve onlara bakarak daha başarılı kararlar verecek.
- İkinci artı şu, end-to-end approach bariz şekilde daha az çaba gerektiyor, bir sürü ayrı problemle uğraşmak yerine tek bir DL modeli fit etmekle uğraşıyoruz.

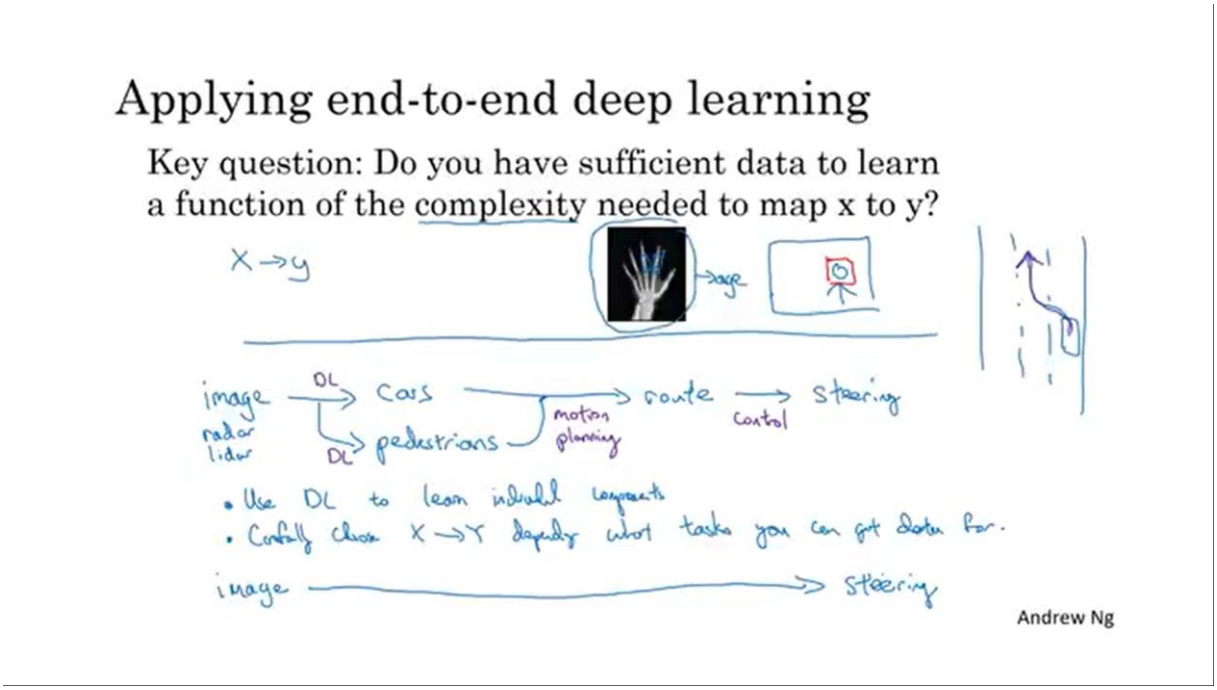


- Eksilerinden birine bakarsak, end-to-end approach için büyük miktarda veri gerekli olabilir.
- İkinci olarak da model bizim sahip olduğumuz intuition'dan mahrum bırakılmış oluyor, yani bu hand-designed features two edged sword. Eğer yeterli data yoksa, model bizim sahip olduğumuz intuiton'dan mahrum kalarak daha muğlak feature'ları tespit etmiş olabilir, oysa biz gidip modele neye bakması gerektiğini söylersek model 5-0 önden başlayacak. Tabi dediğim gibi bu hem bir artı hem de eksi olabilir.

Yani eğer modele yeteri kadar data besliyorsak, hand-design features'a ihtiyacı kalmadan gayet iyi öğrenebilir, ancak yeterli data yoksa hand-designed features model için bir booster görevi görür ve data eksikliğini aslında insan eliyle kapatmış olur, bir nevi biz ona daha eğitime başlamadan önce hangi feature'ların önemli olduğunu söylemiş oluruz.

Sonuç olarak eğer end-to-end deep learning uygulamak ile problemi daha küçük problemlere ayırmak arasında kalmışsak, key question şu: gerekli komplekslikte bir problemi öğrenmeye yetecek kadar datamız var mı? Eğer yeterli data varsa model ham input ile de gayet güzel öğrenecektir, ancak yoksa biz problemi insan eliyle model için daha kolay işlenebilir parçalara böleriz.

Tabi burada gerekli data sayısı problemden probleme değişir ve kesin bir rakamdan söz edilemez. Sadece face detection için çok büyük verilere ihtiyaç yok gibi duruyor görece kolay bir problem ama x-ray fotoğrafından age tahmini yapmak çok daha karmaşık bir problem, bu yüzden end-to-end approach için daha fazla dataya ihtiyacımız olacaktır.



Son olarak autonomous driving örneği verelim. Kendi kendini süren bir arabayı nasıl build ederiz? End-to-end olmayan yakaşım şöyle olabilir:

- Önce radar,lidar vey benzeri sensörlerden alınan veriyi alırız, buna image diyelim.
- Daha sonra bu inputlardan car, pedestrian vb. detection yaparız. Artık önümüzde arkamızda ne var ne yok bunun bilgisine sahibiz.
- Daha sonra arabaya bir route/path lazım. Bunu elde ederiz. (Bu motion planning problemi genelde deep learning ile yapılmaz.)
- Son olarak da bu route'u gerçekleyecek steering, acceleration, breaking commands gerçekleştirilir. (Bu bir control problemi)

En nihayetinde burada DL modeli sadece problemin çözümü için gerekli bazı componentleri elde etmek için kullanıldı.

Buna karşın, eğer elimizde yeterli veri olursa veya belki daha başarılı modeler olursa birgün doğrudan image'a bakarak steering komutlarını veren bir model geliştirilebilir. Ancak şu anda bu yöntem çalışmaz, bunun yerine step by step approach kullanılır.