

COURSE3 – W1

Vid 1

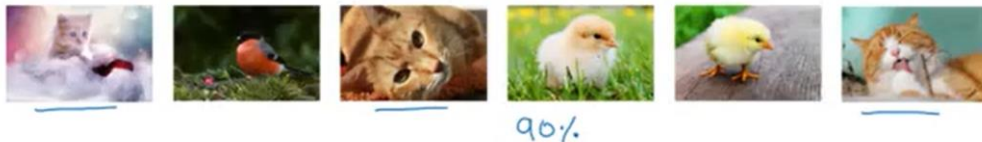
Improving Model Performance

Kurs 3 boyunca “How to structure ML Projects” başlığının altını dolduracağız.

Motivating example olarak şunu kullanalım, diyelim ki bir cat classifier yapıyoruz ve sonuçta model performansı %90 olarak elde ediliyor. Ancak bu bizim amacımız için yeterli değil bu performansı iyileştirmek istiyoruz.

Peki bu durumda ne yapmamız gerekiyor? Aşağıdaki gibi birçok farklı şey deneyebiliriz:

Motivating example



Ideas:

- Collect more data
- Collect more diverse training set
- Train algorithm longer with gradient descent
- Try Adam instead of gradient descent
- Try bigger network
- Try smaller network
- Try dropout
- Add L_2 regularization
- Network architecture
 - Activation functions
 - # hidden units
 - ...

Andrew Ng

Modelimizi geliştirmek için hangisini/hangilerini denemeliyiz? Çünkü yanlış seçimler yapmak, hiç işe yaramayacak bir yolda uzun süre kaynak harcamak anlamına gelir.

İşte bu kursta modelimizi geliştirmek için sistematik bir yol geliştirmek istiyoruz. Bu yollar, deep learning era ile değişiyor, eski ML yöntemlerine göre daha farklı.

Vid 2

Orthogonalization

Bir ML modeli geliştirirken karşımıza çıkan problemlerden biri, deneyecek çok fazla parametrenin olması including so many hyperparameters to tune.

Bu işte iyi olan insanlar, hangi etkiyi sağlamak için neyi tune etmek gerektiğini iyi bilen insanlardır. Bu konseptte orthogonalization denir.

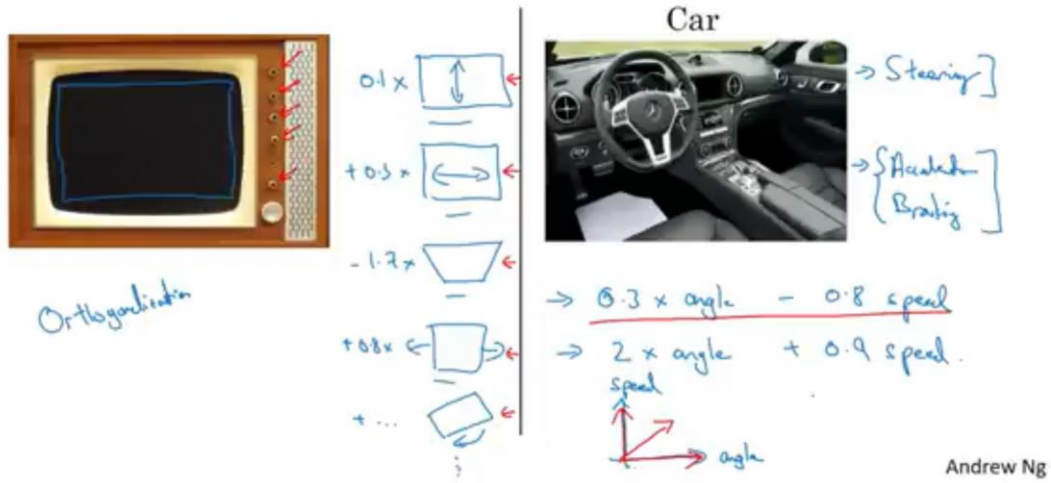
Bu konsepti daha iyi anlamak için iki örnek verelim. Diyelim ki aşağıdaki gibi eski bir TV'miz var, üzerindeki knoblar ile çeşitli ayarlar yapılıyor mesela bir tanesi adjusts how tall is the image, diğeri width'i bir başkası trapezoidallığı kontrol ediyor vesaire.

Buna karşıt olarak şöyle bir durum düşün, bir knob var ve onunla oynayınca tüm ayarlar değişiyor mesela 0.1 height, 0.3 width, -1.7 trapezoidal, +0.8 horizontal position vesaire. Böyle bir knob ile tv'yi istediğimiz gibi tune etmek neredeyse imkansız olur.

Bu bağlamda orthogonalization dediğimizde her knob'un farklı bir ayarı kontrol etmesini yani her ayarın birbirinden bağımsız etkileri olmasından bahsediyoruz. Böylece istediğimiz ayarı yapmak için hangi knob ile oynayacağımızı biliriz ve bu ayarı yaparken bir başka ayarı bozmayız.

Benzer şekilde bir arabanın temel kontrollerini steering ve speed olarak düşünebiliriz, diyelim ki arabayı iki sensörle kontrol ediyoruz bir tanesinin çıktısında $0.3angle$, $-0.8speed$ değişiyor diğeriyle $2angle+0.9speed$ bu şekilde teoride araba istediğimiz gibi kontrol edilebilir ama bu işleri çok daha karmaşık bir hale getiriyor, ayrıca sadece sağa dönmek istediğimde veya yavaşlamak istediğimde bunu tek bir sensörle(mesela direksiyon veya fren) ile yapamayacağım da ikisini birden ayarlamaya çalışacağım bu işleri çok karıştırıyor. Bunun yerine benim istediğim iki bağımsız sensör olmalı (biri speed biri angle'ı kontrol edecek). Bu ikisini bağımsız eksenler gibi düşündüğümüz için bu konseptte orthogonalization diyoruz, eğer bir parametreyi değiştirdiğimizde bundan birden fazla sonuç etkileniyorsa bunu istemeyiz bu işleri karmaşıktırır, ve tuning'i zorlaştırır.

TV tuning example



Peki bu konsepti ML ile nasıl bağlarız?

Bir supervised learning sistemin iyi performans sergilemesi için genelde sistemin knob'ları ile oynayarak 4 şeyi başarmaya çalışırız:

- Modelin önce training set üzerinde iyi çalışması (human level perf. amaçlanır)
- Daha sonra modelin dev set üzerinde iyi çalışması.
- Daha sonra modelin test set üzerinde iyi çalışması.
- Son olarak modelin gerçek problem üzerinde iyi çalışması.

Bu analogide TV'miz ML modele karşılık geliyor, knob'lar modelimiz üzerinde yaptığımız bazı değişiklikler ve TV'mizde image'ı istediğimiz gibi görmek de modelin istenilen performansı sergilemesine denk geliyor.

Peki bu örnekte knob'lar nelere karşılık geliyor?

- Model training set'e iyi oturmuyorsa: Bigger network veya better optimization algorithm kullanmak işe yarar, daha başka örnekleri de daha sonra göreceğiz. Daha uzun eğitim de mantıklı.
- Eğer algoritma train set üzerinde iyi ama dev set üzerinde kötü performans veriyorsa (overfitting), bunun için başka knoblar kullanılır: Regularization veya getting a bigger training set.
- Eğer test set üzerinde kötü performans sergileniyorsa: Bigger dev set. Çünkü dev set üzerinde iyi performans aldıysak ama test set üzerinde alamadıysak modeli dev set üzerinde overtune ettik demektir. Aslında bu aynı overfit etmek gibi, sadece overfitting training set üzerinde oluyor çünkü model parametreleri training set kullanılarak fit ediliyor, hyperparameters ise dev set kullanılarak fit ediliyor bu yüzden overtune da dev set üzerinde oluyor, bu durumda bigger dev set işe yarar.
- Eğer test set üzerinde iyi performans aldıysak ama model gerçek hayatta problemi çözmekte yeterli performansı sergileyemiyorsa: Geri dönmeli ve ya test

set'i (videoda dev demiş ama bence test set olmalı yanlış söyledi.) ya da cost function'ı değiştirmeliyiz. Çünkü eğer model test set üzerinde iyiye ama gerçekte kötüye demekki test set gerçek hayat dataset ile uyuşmuyor veya, cost function aslında bizim istediğimiz performans kriteriyle uyuşmuyor.

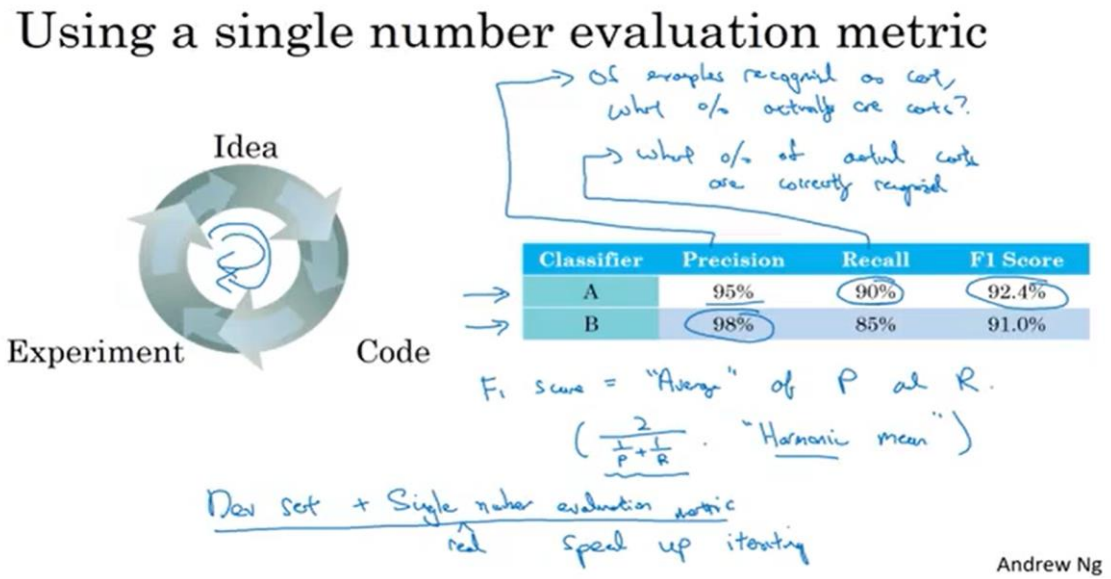
Son olarak Andrew kendisinin "early stopping" tekniğini kullanmaktan kaçındığından bahsediyor. Bir çok insan kullanıyor, işe de yarıyor ancak orthogonalization prensibine uymuyor yani early stopping öyle bir knob ki birden fazla ayarı etkiliyor. Bu genelde dev set performansını artırmak için (overfitting engellemek için) kullanılıyor, ancak bunun yanında training set performansını da düşürüyor, çünkü eğitimi erken durdurmuş oluyoruz. Kullanılmaz değil kullanılabilir ancak, diğer orthogonal knob'lar varken bunu yapmak işlerin kontrolünü biraz kaybetmemiz anlamına gelebilir aynı araba örneği gibi.

Vid 3

Single Number Evaluation Metric

İster hyper parameter tune ediyor olalım ister learning algorithm için farklı fikirler deniyor olalım eğer single real number evaluation metric kullanırsak işlerin çok daha hızlı ilerlediğini görürüz, böylece bir fikri veya parametreyi denediğimizde işe yarayıp yaramadığını rahatlıkla söyleyebiliriz.

Aşağıda bu durum için bir örneğe bakalım:



Daha önce de söylendiği gibi bir ML modeli geliştirme süreci iteratif bir süreç, sürekli bir şeyler deneyip performansa bakılır ve daha iyileştirme için süreç tekrarlanır.

Diyelim ki bir classifier A eğittik, ve bir şeyleri (hyperparameters veya başka bir şey) değiştirerek bir başka classifier B'yi eğittik.

Modellerimizin performansını evaluate etmenin makul yollarından biri precision ve recall kavramlarından yararlanmak. Precision dediğimiz şey, cat olarak classify edilenlerin yüzde kaç gerçekten kedi. Recall da gerçek kedilerin yüzde kaçının doğru bilindiği. Bu tanımların şuan çok önemi yok.

Genelde precision ile recall arasında bir tradeoff söz konusudur, bu yüzden birisini daha iyi yapan bir classifier diğerinden ödün veriyor demektir, yukarıda da A'nın recall için B'nin ise precision için daha iyi olduğunu görürüz.

Biz birini diğerine tercih etmeyiz, ikisi de önemlidir. Diyelim ki 10 farklı classifier'ımız var, hepsini farklı hyperparameter ile eğittik, hepsinin farklı precision ve recall değerleri var, hangisini seçeceğimizi bilemeyiz, çünkü tek bir evaluation metrics kullanmıyoruz.

Bu noktada genelde F1 Score kullanılır, bu precision ile recall'ın harmonic ortalamasından fazlası değildir.

Sonuçta iyi bir dev set ile single number evaluation metric kullanılması iteratif model geliştirme sürecini hızlandırır.

Şimdi bir diğer örneğe bakalım:

Another example

Algorithm	US	China	India	Other	Average
A	3%	7%	5%	9%	6%
B	5%	6%	5%	10%	6.5%
C	2%	3%	4%	5%	3.5%
D	5%	8%	7%	2%	5.25%
E	4%	5%	2%	4%	3.75%
F	7%	11%	8%	12%	9.5%

Andrew Ng

Diyelim ki biz bir startup'ız ve app ile kullanıcıların yüklediği kedileri classify ediyoruz, farklı classifiers farklı bölgelerdeki kullanıcıların yüklediği fotoğraflar üzerinde farklı hatalar veriyor. E hangisini seçeceğiz? Olay yine single number evaluation metric kullanmakta bitiyor, bu yüzden hataların average'ını kullanabiliriz bu durumda en mantıklısı E modelini kullanmak olacak.

Vid 4

Satisficing and Optimizing Metrics

Önceki kısımda modellerin farklı performans kriterlerini ayrı ayrı değerlendirerek hangisinin daha iyi olduğunu karar vermenin zor olduğundan bu yüzden de tüm kriterlerin performansını içeren bir nevi average performans kriterine yani single number evaluation metric'e ihtiyacımız olduğunu söylemiştik.

Ancak farklı performans kriterlerini tek bir evaluation metric ile ölçmek her zaman mümkün olmayabilir. Yani modelin iyi yapmasını istediğim şeyleri tek bir evaluation metric ile göstermek her zaman kolay olmaz.

Bu durumda satisficing ve optimizing metrics kavramları devreye girer.

Another cat classification example

Classifier	Accuracy	Running time
A	90%	80ms
B	92%	95ms
C	95%	1,500ms

Handwritten notes:

Optimizing (pointing to Accuracy)
Satisficing (pointing to Running time)

Wakewords / Trigger words
Alexa, OK Google,
Hey Siri, nihao baidu
你好 百度

Accuracy.
#false positive

Maximize accuracy.
s.t. ≤ 1 false positive
every 24 hours.

Cost = accuracy - 0.5 * Running Time

Maximize accuracy
Subject to Running Time ≤ 100 ms.

N metrics: 1 optimizing
N-1 satisficing

Andrew Ng

Diyelim ki elimizde farklı classifiers var ve accuracy için precision recall gibi farklı faktörlerin hepsini gözeterek bir accuracy metric kullanıyoruz örneğin F1 Score. Ancak bize sadece accurate bir model değil bunun yanında hızlı çalışan bir model gerekiyor.

Bu durumda yapılabilecek şeylerden biri accuracy ve running time'ı gözetken combined bir evaluation metric üretmek, mesela $cost = accuracy - 0.5 * running\ time$ ancak bu biraz suni görünüyor.

Buna bir alternative olarak yapılabilecek bir şey şu: running time $< 100ms$ olan modellerin arasından accuracy'yi maximize eden modeli seç. Bu durumda accuracy is an optimizing metric çünkü bunu maximize etmeye çalışıyoruz, ama running time bir satisficing metric yani bunun belli bir değer aralığında olmasını istiyoruz o kadar bir optimizasyon söz konusu değil.

Böyle bir evaluation durumunda classifier B'nin performansı daha iyi diyebiliriz.

Daha genel bir ifadeyle eğer elimizde N metric varsa, bunlardan bir tanesini optimizing diğer N-1 tanesini satisficing metric olarak seçmek mantıklı olacaktır.

Bir başka örnekten bahsedelim, mesela wakewords veya triggerwords'u detect eden bir system geliřtiriyoruz, örneğın Hey Siri deyince system uyanacak, bu tespiti yapabilen bir classifier eğittik diyelim.

Bu systemin performansını değerdendirmek için, accuracy yani Hey Siri denildiğında hangi kesinlikle yakalıyor olduğı önemli ancak bunun yanında false positives de önemli olabilir yani Hey Siri denmediğında hangi sıklıkla system randomly wakes up.

Bu durumda bu iki evaluation metricsi combine etmenin bir yolu false positivesi satisficing metric olarak seçip, accuracy maximize etmek, yani günde en fazla 1 false positive veren max accuracy modeli ararız.

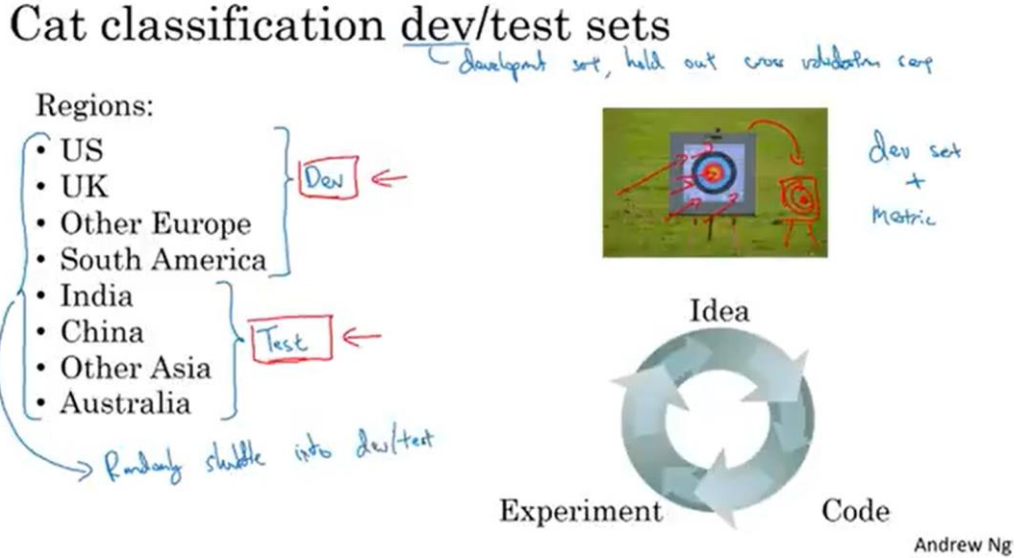
Bu yöntemle birden fazla classifier'a bakıp hangisinin daha iyi performans sergilediğini rahatlıkla belirleyebiliriz.

Vid 5

Train/Dev/Test Set Distributions

Bu setleri belirlemek, hızlı yol alabilmek için çok önemli. Verimliliği maximize etmek için bu setleri nasıl oluşturacağımıza bakalım.

Bu kısımda Dev ve Test setlerini nasıl oluşturacağımızdan bahsedeceğiz.



Önceki örneği kullanalım kullanıcıların kedi fotoğraflarını classify eden bir model build ediyoruz diyelim. Elimizde farklı regionlardan alınmış kedi fotoğrafları var. Bu verinin yarısını Dev set Yarısını Test set yağıcağız diyelim. Datayı nasıl ayırmalıyız?

Bir olasılık şu: yukarıdaki gibi 4 ülkenin verilerini dev set yaparız, diğer 4 ülkeninkileri test set yaparız. Bu çok kötü bir fikir, çünkü bu durumda dev ve test set farklı dağılımlardan gelir.

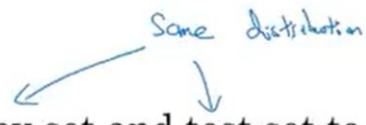
Yani belki çinliler sürekli tombiş kedi fotoğrafları yüklerken, avrupalı kullanıcılar sokak kedisi fotoğrafları yüklüyor, bu durumda biz modelimizi sokak kedilerine göre tune edeceğiz ve bu data üzerinde iyi sonuçlar alacağız ancak test set üzerinde performans değerlendirmesi yapınca model çuvalalayacak çünkü aslında eğitme gördüğü şey bu değildi.

Bu yüzden yapılacak doğru şey, her bölgeden gelen verileri shuffle edip bölmektir böylece dev ve test setleri benzer distributionlar sergileyecek ve modelimizin dev set üzerinde tune edilince test üzerindede iyi performans sergileyecek.

Yani bir dev set ve evaluation metric belirlediğimizde aslında bir hedef belirlemiş oluruz, modelin bu distribution üzerinde bu metriğe göre başarılı olması için gerekli parametreleri ararız ve buluruz eğer modelin test distribution farklıysa önceki çalışmaların bir önemi kalmaz çünkü modeli bambaşka bir şey için hazırlamışız demektir.

Guideline

Choose a dev set and test set to reflect data you expect to get in the future and consider important to do well on.



Andrew Ng

Yani modelin nasıl bir data üzerinde çalışmasını istiyorsak, bu data içinden aynı distribution'a sahip bir veri seti alıp bunu da yine distribution koruyacak şekilde dev ve test set'e ayırmamız gerek. Böylece eğitilen model gerçekte de iyi performans sergileyecektir.

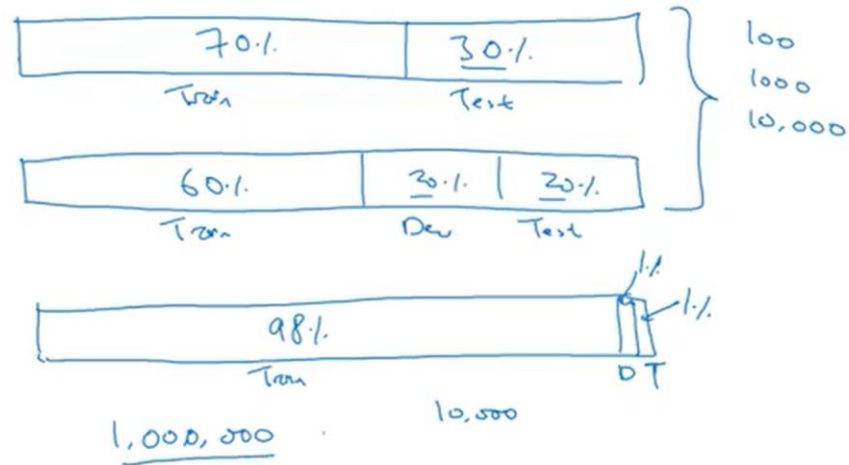
Bu kısımda training setten bahsetmedik, daha sonra bahsedilecek. Bununla ilgili şunu söyleyebiliriz dev ve test setleri ile modelimizin amacını ortaya koyarız yani modelimizin hangi veri üzerinde çalışmasını istiyorsak dev ve test set ona göre aynı distributiondan seçilmeli, training set seçimi ise bu hedefte ne kadar iyi performans sergileyebiliriz onunla ilgili. Bunu daha sonra göreceğiz.

Vid 6

Size of Dev and Test Sets

Önceki kısımda dev ve test setlerinin aynı distributiondan gelmiş olması gerektiğini söylemiştik. Dev ve Test set boyutlarının seçimi de modelimizin performansını etkileyen etkenlerden, deep learning era da boyut seçimi de biraz değişti.

Old way of splitting data



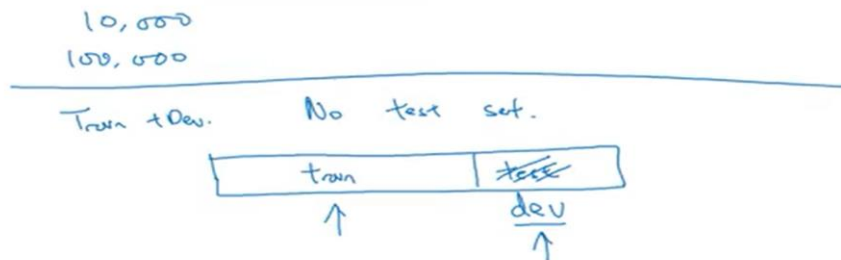
Andrew Ng

Eskiden dağılımı yukarıdaki gibi 70% - 30% veya 60%-20%-20% şeklinde yapıyorduk. Eğer elimizde 10 000 civarı örnek varsa dataseti böyle bölmek hala mantıklıdır.

Ancak deep learning era'da çok genelde çok daha büyük datasets ile uğraşırız, diyelim ki 1000000 datamız var bu durumda 10 000 dev 10 000 test set olarak bölmek mantıklı yani bu örnekte bu %1'e denk geliyor, deep learning modelleri dataya aç oldukları için datanın büyük bölümünü training için kullanırız.

Size of test set

→ Set your test set to be big enough to give high confidence in the overall performance of your system.



Andrew Ng

Peki ya test set boyutu? Test setinin amacı modelin gelişimini tamamladıktan sonra modelimizin ne kadar iyi performans sergilediğini unbiased bir şekilde evaluate etmemize yarar.

Bu bağlamda test set'in real problem'i high confidence ile temsil etmesi yeterli, eğer modelin gerçek hayat performansını küsüratlarına kadar kesin olarak istemiyorsak, öyle çok fazla örneğe ihtiyacımız yok, genel hatlarıyla real distribution'ı bozmayan bir set ayırılım yeter.

Bazı uygulamalarda, modelin overall performansı için high confidence'a ihtiyacım olmayabilir, bu yüzden sadece train ve dev setleri kullanılabilir. Kimisi buna train-test set der bu çok doğru bir terminoloji kullanımı değil.

Andrew test set'in olmamasını önermiyor, modelin performansının unbiased bir estimate'ı için bir test set kullanmayı mantıklı görüyor. Ancak eğer dev setimiz yeterince büyükse ve modeli dev set üzerinde overtune etmeyeceğimizi düşünüyorsak, belki test set kullanmasak da olabilir.

Sonuçta big data erada eski ML split kuralları artık işlemiyor, onun yerine artık training için daha fazla data kullanmak popüler. Rule of thumb şu: dev seti ve test seti kendi amaçları için yeterli büyüklükte tutsak yeter bunun için %20 ayırmaya gerek olmayabilir. Dev set'in amacı: help you evaluate different ideas and pick one. Test set'in amacı: help you evaluate your final classifiers.

Vid 7

When to Change Dev/Test Sets

Daha önce kurduğumuz bir analogi ile, dev set + evaluation metric belirleyerek aslında modelimize bir target belirlediğimizden bahsetmiştik. Bazen model geliştirmenin ortasında farkederiz ki target'i yanlış yere koymuşuz yani modelimiz yanlış target'i hedefliyor, bu durumda target'i doğru yerine çekmeliyiz. Bunu örnekle daha iyi anlayalım.

Cat dataset examples

Metric + Dev : Prefer A
You/users : Prefer B.

→ Metric: classification error

Algorithm A: 3% error → pornographic

✓ Algorithm B: 5% error

$$\left\{ \begin{array}{l} \text{Error: } \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{\text{dev}}} w^{(i)} \mathbb{I}\{y_{\text{pred}}^{(i)} \neq y^{(i)}\} \\ \rightarrow w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-porn} \\ 10 & \text{if } x^{(i)} \text{ is porn} \end{cases} \end{array} \right.$$

↗ predicted value (0/1)

Andrew Ng

Diyelim ki cat lover users'ın yüklediği fotoğrafları classify eden bir model inşa ediyoruz. Bir algoritma %3 diğeri ise %5 error veriyor, algorithm A daha iyi gibi görünüyor. Ancak bir sebeple A modeli kedileri %2 daha iyi belirlese de pornografik fotoğrafları elimine edemiyor, ve kullanıcılara pornografik içerikler göstermek zorunda kalıyor, buna karşın B kedileri pek iyi ayırt edemese de pornografik içerikleri A'ya göre daha iyi ayırt ediyor.

Bu sebepten bizim evaluation metriğimiz algorithm A daha iyi dese de, kullanıcının ve bizim gözümüzden bakınca model B çok daha iyi bir model.

Yani burada evaluation metrik+dev set targeti ile bizim targetimiz uyuşmuyor. Bu durumda bizim evaluation metric+dev/test setimizi yani targetimizi değiştirmemiz gerek.

Bu durumda error metriğimiz yukarıda mavi ile görünen kısım gibi olabilir, bu formül temelde yanlış classify edilen örnekleri sayıyor. Ancak bu evaluation metriğin problemi pornografik ve pornografik olmayan örnekleri aynı kabul etmesi halbuki bizim için pornografik örneği geçirmesi çok büyük problem, bu sebeple bu metriği modify edebiliriz, basitçe bu metriğe bir weight ekliyoruz ve eğer ilgili örnek pornografik ise sonuçta oluşacak error term 10 kat daha büyük olsun deriz. Error 0-1 arasında olsun istersek başına da bir normalizasyon faktörü ekleriz.

Tabii ki bu weighting'in yapılabilmesi için dev ve test set üzerinde pornographic örneklerin labellenması lazım.

High level takeaway is: Eğer elimizdeki evaluation metriğin seçtiği en iyi algoritma ile bizim isteklerimiz arasında uyumsuzluk varsa, o zaman evaluation metriği değiştirmeliyiz, yukarı açıklanan yöntem bu metriği değiştirmenin sadece bir yolu.

Evaluation metriğin amacı, bize hangi modelin daha iyi olduğunu kesin bir şekilde söylemektir.

Şuana kadar sadece bize modellerin performansını veren bir evaluation metriği nasıl define edebiliriz onunla uğraştık, aslında bu da bir orthogonalization örneği.

Orthogonalization for cat pictures: anti-porn

- 1. So far we've only discussed how to define a metric to evaluate classifiers. ← Place target \mathcal{J}_0
- 2. Worry separately about how to do well on this metric. \mathcal{J}_1

$$\mathcal{J} = \frac{1}{\sum w_i} \sum_{i=1}^m w_i \ell(\hat{y}^{(i)}, y^{(i)})$$

\uparrow Aim (shoot at target)



Andrew Ng

Bir knob ile, bizim isteklerimizi gözetken bir evaluation metriği nasıl tanımlarız onu control ediyoruz.

Bu metriği kendine hedef edinen modelin, bu hedefi ne kadar iyi yakalayıp yakalamadığı bağımsız bir knob ile control edilmeli.

Yani ML taski iki orthogonal step ile düşünebiliriz, önce hedefimizi yerleştirmeliyiz, tam olarak hangi kriterde iyi olan bir model istediğimizi belirlemeliyiz yani metriğimizi belirlemeyiz. Daha sonra bağımsız bir knob ile bu hedefi ne kadar başarılı bir şekilde tutturabiliriz onu control etmeliyiz.

Son bir örneğe daha bakalım:

Diyelim ki kedi classify eden 2 modelimiz var: A ve B 'nin dev veya test set errorları %3 ve %5 olarak hesaplanmış. Ancak gerçek hayatta iki modeli kullandığımızda bir de bakıyoruz ki B daha iyi performans veriyor. Çok geçmeden bunun sebebinin modelin yüksek kalite fotoğraflarla eğitilmiş olmasına karşın, kullanıcıların yüklediği fotoğrafların düşük kaliteli olması olduğunu anlıyoruz.

Bu da bizim metric + dev/test setimizin yani targetimizin problemlili olduğunu gösteriyor, bu durumda yapmamız gereken metriğimiz and/or dev/test setimizi değiştirmek!

Another example

Algorithm A: 3% error

✓ Algorithm B: 5% error ←

→ Dev/test



→ User images



If doing well on your metric + dev/test set does not correspond to doing well on your application, change your metric and/or dev/test set.

Andrew Ng

Vid 8

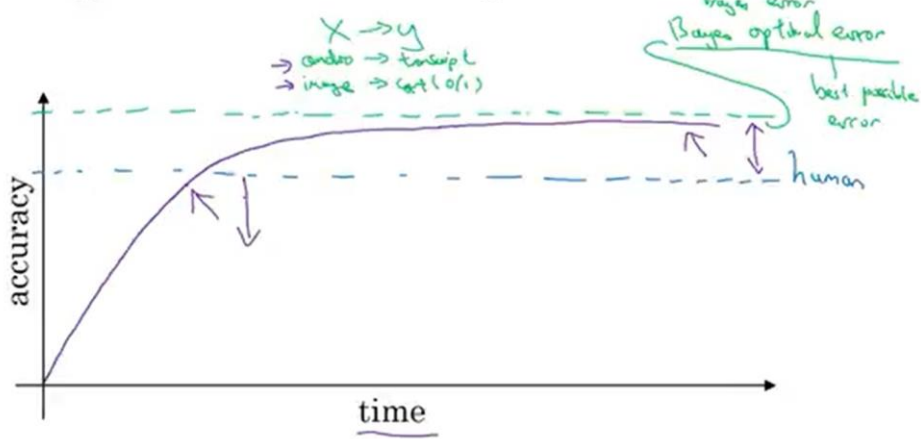
Why Human Level Performance

Son yıllarda bir çok ML takımı, ML sistemlerini “human level performance” ile karşılaştırıyor. Bunun iki nedeni var:

Birincisi DL gelişmeleri ile artık ML algoritmaları birçok alanda insanlar ile rekabet edebilecek bir düzeye ulaştı. İkinci sebep ise, ML algoritmaları ile insanların yapabileceği işler yapılmaya çalışıldığında algoritmanın design ve built workflowları daha verimli oluyor.

Birkaç örnekle durumu daha iyi kavramaya çalışalım.

Comparing to human-level performance



Genel algoritma performansları yukarıdaki mor eğri gibi bir yol izler, time eksenini bir takımın model üzerinde çalışma zamanı olarak düşünebiliriz, yani modelin accuracy'si human level performance seviyelerine gelinceye kadar gayet hızlı yükselir ancak daha sonra model iyileştirilmeye devam edilse de hızı zamanla bariz şekilde yavaşlar.

Model geliştirilmeye devam edildikçe, hopefully performance approaches to a theoretical limit (green line). Bu limiti belirleyen şey “Bayes Optimal Error” olarak adlandırılır, bunu best possible error olarak düşün. X'i y'ye map eden hiçbir fonksiyon bundan daha az bir errorla map edemez. Yani diyelim ki speec recognition yapıyoruz, ancak bazı örnekler o kadar noisy ki bunu ayırt etmek imkansız, veya image recognition yapıyoruz bazı resimler çok blurry, bu örnekler ayırt edilemez, öyle bir fonksiyon yok. Bu yüzden perfect accuracy %100 olmayabilir bunun yerine bayes optimal error'un izin verdiği accuracy olabilir.

Bu yüzden yukarıdaki grafikte model accuracy'si yüzyıllarca model geliştirilse de yeşil limiti aşamaz, çünkü evrende modelin aradığı gibi bir fonksiyon yok.

Model accuracy artış hızının human level performance'ı geçince yavaşlamasının iki sebebi var:

Birincisi şu: insanların performansı çok iyi, yani bayes optimal error'a çok yakınlar, bir fotoğrafa bakınca kedi mi değil mi çok iyi bir şekilde anlayabiliyor, yani genelde bir model human level performance'a gelince zaten üst limite çok yakın demektir dolayısıyla artış hızının yavaşlaması çok normal.

İkincisi ise şu: model performansı human level performance'ın altında ise performansı yükseltmek için kullanabileceğimiz bazı toollar vardır bu tooları human level performance üstünde kullanamayız bu yüzden hlp seviyesine kadar model performansını hızlı bir şekilde ilerletebiliriz.

Nedir bu toollar?

Why compare to human-level performance

Humans are quite good at a lot of tasks. So long as ML is worse than humans, you can:

- Get labeled data from humans. (x, y)
- Gain insight from manual error analysis:
Why did a person get this right?
- Better analysis of bias/variance.

Andrew Ng

Eğer model performansı insanlardan kötüyse, insanlardan labelad data elde edebilirim ve böylece modelim için daha çok verim olacak.

Ayrıca gelecek hafta detaylı göreceğimiz, manual error analysis yapabiliriz elimizde iyi performansın bir örneği olduğu için (insanlar) neden modelin yanlış fakat insanların doğru yaptığına dair bir insight edinebiliriz.

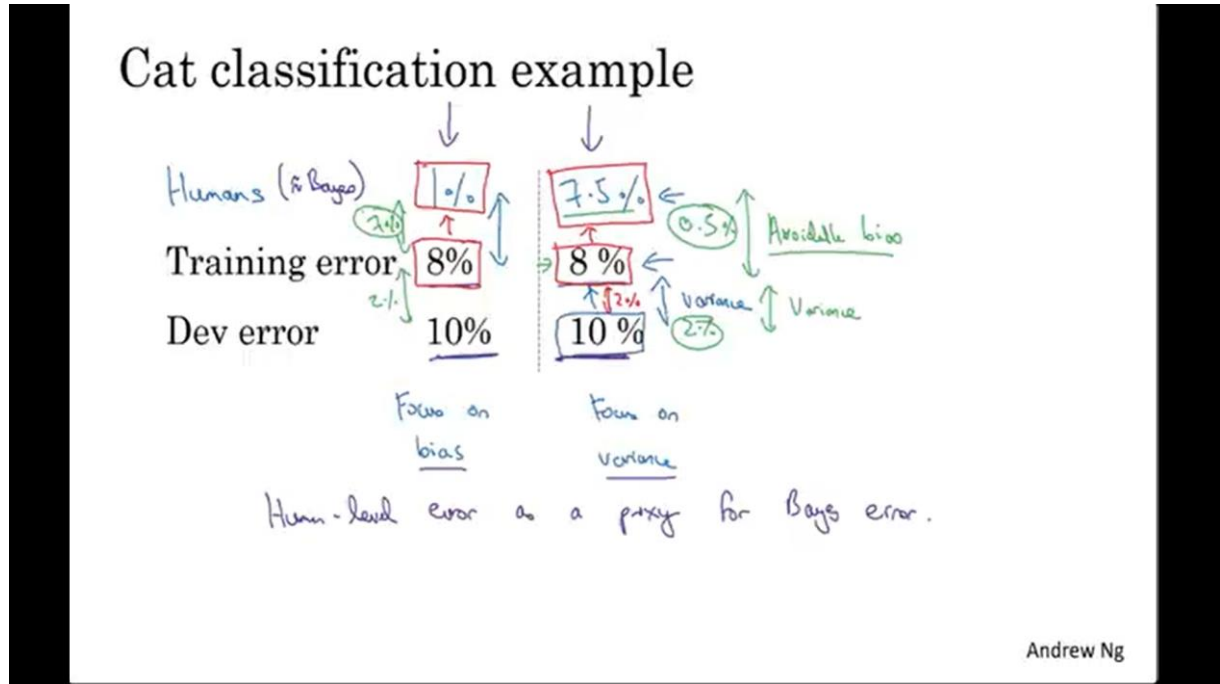
Ayrıca yine bir sonraki kısımlarda göreceğiz, daha iyi bir şekilde bias/variance analizi yapabiliriz.

Bu toollar sayesinde, ML modelleri human level performansı hızlı bir şekilde yakalayabilir belki biraz da geçebilir.

Bir sonraki kısımda bir task üzerinde human level performans'ı biliyor olmanın, bias ve variance'ı ne kadar reduce edeceğimize karar vermemizde nasıl etkisi olduğunu göreceğiz.

Vid 9

Avoidable Bias



Burada açıklanan şu: human level performans bize bias ve variance düzenlemesinde yol gösterir.

Eğer bir problem için human level performance: %1 hata iken training ve dev error %8 ve %10 ise modelin bias'ının yüksek olduğunu söyleyebiliriz, bu yüzden bunu düşürmek için belki daha büyük bir network veya optimization algorithm kullanmalıyız veya belki de eğitim süresini uzatmalıyız.

Ancak başka bir örnek için aynı dev ve test error veren bir model için başka bir yaklaşım uygulamamız gerekebilir, çünkü belki o örnek için human level performance 7.5% error veriyor öyleyse %8 training error'a acceptable diyebiliriz ve training error'u düşürmeye uğraşmak yerine, regularization veya daha fazla training data ile dev error'u düşürmeye çalışabiliriz.

Genel olarak human level performance hatasını Bayes Optimal Error olarak Kabul edebiliriz elbette tam olarak böyle değildir ama insanlar genelde minimum olası hatadan birazcık daha fazla hata yaparlar. (Cat classification vb. gibi iyi oldukları konulardan bahsediyoruz.)

Sonuçta modelin training error'u ile human level performance arasındaki fark'ı "Avoidable Bias"ın bir ölçüsü olarak düşünebiliriz, çünkü hlp'ı aşağı yukarı bayes optimal error kabul edersek, training error ile hlp arasındaki fark bize önlenebilecek bir bias gösterir, bu fark ne kadar az ise bunu önlemek için çabalamak o kadar boş çünkü zaten daha iyisi yok. Dev error ile training error farkı ise variance'ın bir göstergesi.

Bu yüzden hlp'yi bilmek önemli ona göre önce bias'e mi yoksa variance'a mı odaklanmalıyız, bias'e odaklanırsak hedefimiz ne olmalı gibi soruları cevaplamamızı sağlar.

Vid 10

Understanding Human-Level Performance

Bazen “Human Level Performance” lafı gelişigüzel kullanılabiliyor, bu tanıma biraz daha yakından bakalım.

Daha önceki kısımdan hatırlayacağımız üzere HLP’yi Bayes Error’un bir estimate’ı olarak kullanabiliyorduk.

Şimdi bir örneğe bakalım, diyelim ki aşağıdaki gibi bir radiology image’ına bakıp diagnosis koymak istiyoruz. Bu task için aşağıda farklı grupların hata yüzdeleri verilmiş.

Human-level error as a proxy for Bayes error

Medical image classification example:

Suppose:

- (a) Typical human 3 % error
- (b) Typical doctor 1 % error
- (c) Experienced doctor 0.7 % error
- (d) Team of experienced doctors .. 0.5 % error ←



What is “human-level” error?

Bayes error \leq 0.5%

Andrew Ng

Bu durumda HLP’yi nasıl define edeceğiz? HLP’yi her zaman Bayes Error’un estimate’ı olarak düşünmekte fayda var bu sebeple, eğer team of experience doctors 0.5% hata yapıyorsa, bayes error bundan daha düşük demektir, biz de bayes error’un 0.5%’den çok daha az olmadığı kabulüyle HLP’yi %0.5 olarak kabul edebiliriz.

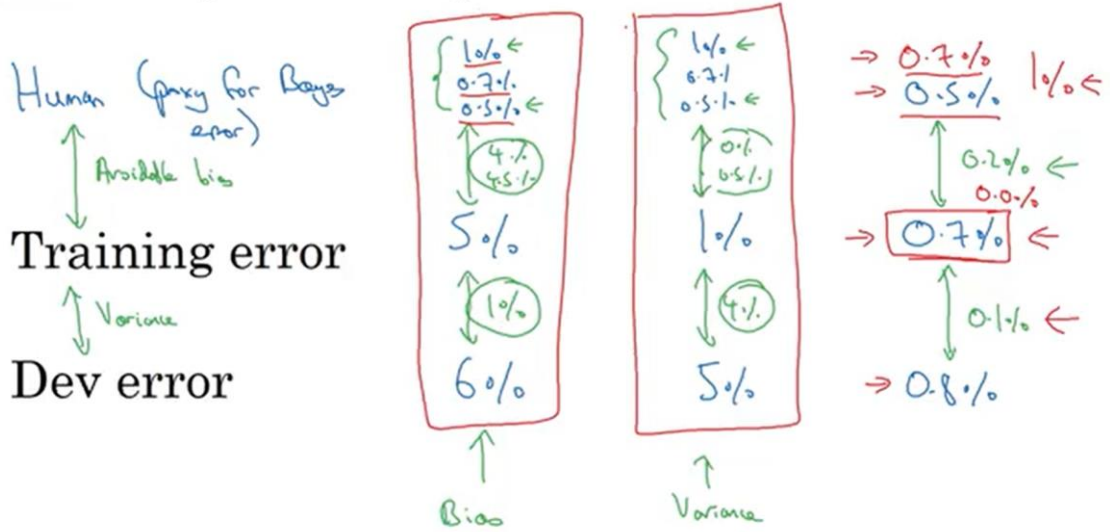
Ancak modelin kullanımına bağlı olarak, 1% i de HLP olarak alabiliriz.

Şimdi aşağıda bir error analysis örneğine bakalım: diyelim ki medical image diagnosis yapıyoruz. Training error 5% dev error %6 olarak tespit edildi.

Yukarıdaki konu bağlamında HLP'yi (estimate for bayes error) modelin amacına göre 1% , 0.7% veya 0.5% gibi bir degree set edebiliriz.

Training error ile HLP arasındaki fark Avoidable Bias'ın bir ölçüsüydü. Benzer şekilde Dev Error ile Training Error farkı da Variance'ın bir ölçüsüydü.

Error analysis example



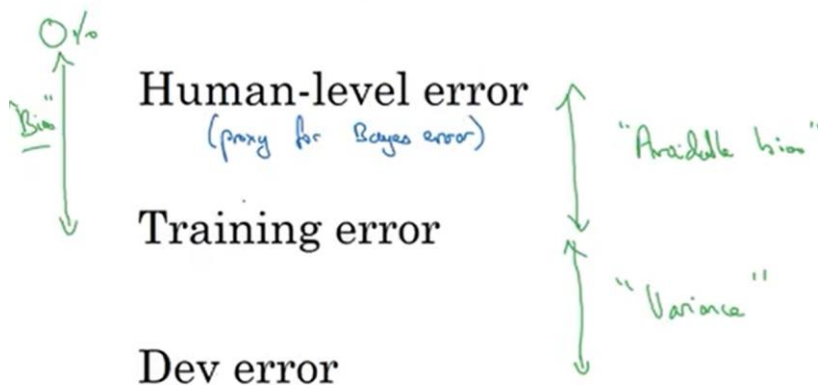
Andrew Ng

Soldaki örnekte hangi HLP'yi seçtiğimiz çok önemli değil, sonuçta %4-4.5 civarı bir avoidable error görüyoruz, buna karşın variance ölçüsü %1 civarında bu durumda açıkça bias reduction techniques üzerine yoğunlaşmalıyız (bigger network, better optimization, longer training).

Ancak ikinci duruma bakarsak, diyelim ki training error %1 olan bir model eğittik bu durumda da HLP'nin hangisi seçildiği önemli değil Avoidable Bias %0-0.5 civarındayken, variance %4 civarında bu durumda bias reduction techniques kullanılmalı (regularization, more training examples)

3. örnekte ise HLP'nin seçimi önem teşkil eder eğer 0.7% seçilirse variance çözülmeye çalışılır ancak 0.5% seçilirse önce bias çözülmeye çalışılır.

Summary of bias/variance with human-level performance



Andrew Ng

Özetle insanların iyi olduğu işler için HLP'yi Bayes Error'un yani olabilecek minimum error'un bir estimate'ı olarak kabul edebiliriz.

HLP ile Training Error arasındaki fark bize avoidable bias ile ilgili fikir verirken, Training Error ile Dev Error farkı bize Variance ile ilgili fikir verir sonuçta biz modelimizde yapacağımız bir sonraki değişikliğe bunlara göre karar veririz.

Yani training error'u 0% ile karşılaştırmak ve aradaki farkı bias olarak görmek doğru değil, çünkü bazen ilgili işlemi hatasız yapmak mümkün olmaz, evrende böyle bir fonksiyon yoktur.

Bu tekniklerin model performansının HLP'yi surpass edene kadar geçerli olduğunu unutma çünkü daha sonra modeli geliştirmek için önümüzde bir Bayes Error yoktur, modelimizin bias problemi mi yoksa variance problemi mi olduğunu anlamak güçleşir.

Vid 11

Surpassing Human-Level Performance

Daha önce ML performans gelişiminin HLP'ye yaklaştıkça veya geçtikten sonra hızla yavaşladığını görmüştük.

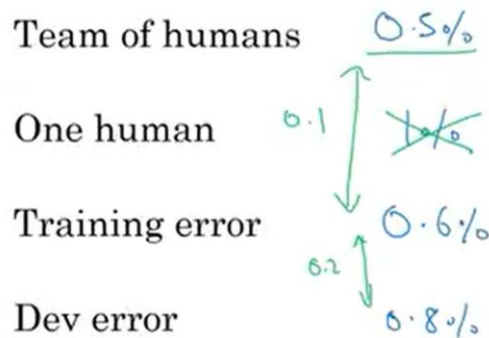
Bunu daha iyi anlamak için bir örnekten daha bahsedelim. Diyelim ki bir problemimiz var team of humans 0.5% error ile problemi çözebilirken, one human 1% ile çözüyor training error 0.6% ve dev error ise 0.8%.

Bu durumda avoidable bias'ımız nedir? Algoritma zaten 1% den iyi yapıyor, o zaman bayes error olarak 0.5% kabul ederiz ve avoidable bias measure'ı 0.1% olarak alırız, buna karşın variance measure ise 0.2% o zaman ilk etapta variance problemini çözmek daha mantıklı.

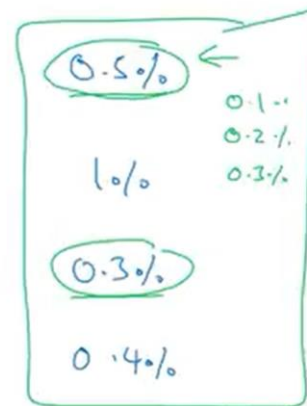
Şimdi yüzde hataların sağ alttaki gibi olduğunu varsayalım bu daha zor bir örnek. Bu durumda işler karışıyor çünkü, training error %3 gelmiş, acaba bayes error 0.1% veya 0.2% veya 0.3% ve biz de human level performance'ı surpass'mi ettik yoksa bayes error 0.5% ve biz training data'ya overfit ettiğimiz için mi 0.3% hata aldık, bunu cevabını bilmiyoruz. Bu yüzden bu bilgilerle bias'ı mı düşürmeliyiz, variance'ı mı bilmiyoruz bu yüzden algoritma gelişimi yavaşlıyor.

Ayrıca algoritma eğer insandan daha iyi performans veriyorsa, artık modeli iyileştirmek için intuitive değişiklikler yapmak da işe yaramayabilir çünkü model o task üzerinde artık bizim anlamadığımız bir şeyleri anlıyor.

Surpassing human-level performance



What is avoidable bias?



Andrew Ng

Birçok problem için ML HLP'yi geçiyor yani insanlardan daha iyi performans sergiliyor. Sol aşağıdaki 4 başlık bunlara örnek, bu problemlerin hiçbiri natural perception(computer vision, speech recognition, NLP) değil, ayrıca hepsi için bir sürü structured data var, hiçbir insan hayatında bu kadar dataya bakmamıştır bu yüzden algoritmanın insanı geçmesi çok tuhaf değil.

İnsanlar Natural Perception Tasks için ML'e göre çok daha başarılı, buna rağmen bazı speech recognition, image recognition veya medical diagnosis uygulamalarında ML sistemleri HLP'yi surpass etmeyi başardı, tabi bunu başarmak soldakilere göre daha zordu.

Problems where ML significantly surpasses human-level performance

- - Online advertising
- - Product recommendations
- - Logistics (predicting transit time)
- - Loan approvals

Structured data
Not natural perception
Lots of data

- Speech recognition
- Some image recognition
- Medical
 - ECG, Skin cancer, ...

Andrew Ng

Vid 12

Improving Model Performance

Bu hafta bir çok başlıktan bahsettik:

- Orthogonalization
- How to set up dev/test sets
- HLP as a proxy for Bayes Error
- How to estimate your avoidable bias and variance

Şimdi bunları birleştirelim ve learning algoritmimizin performansını nasıl geliştireceğimizle ilgili bir guideline oluşturalım.

Bir supervised modelinin iyi olduğunu söylemek için aşağıdakileri yapabiliyor olduğumuzu kabul ediyoruz.

The two fundamental assumptions of supervised learning

1. You can fit the training set pretty well.

~ Avoidable bias

2. The training set performance generalizes pretty well to the dev/test set.

~ Variance

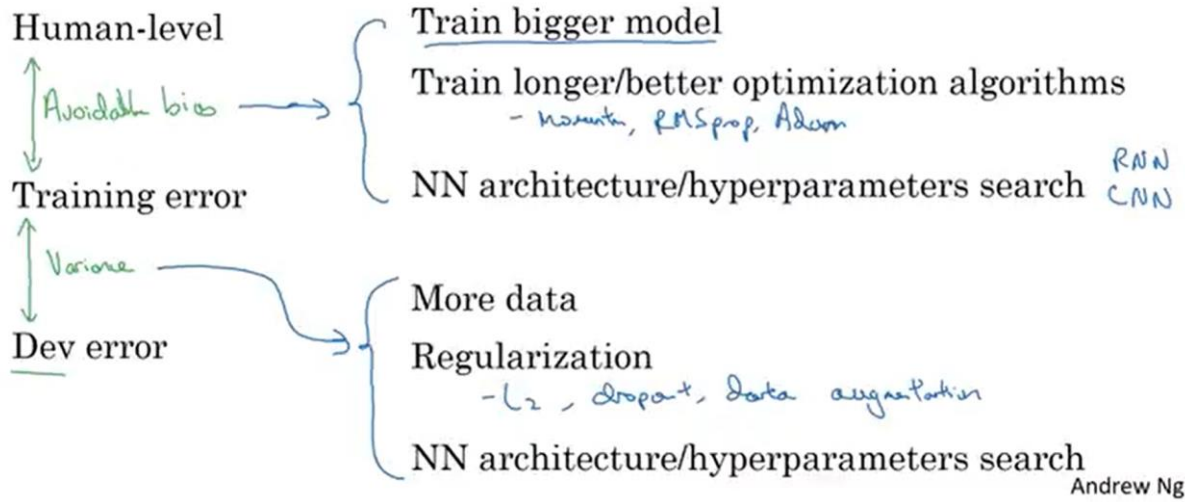
Andrew Ng

Modeli training set'e iyi oturtabiliyoruz, yani düşük bir avoidable bias ve ayrıca düşük bir variance elde edebiliyoruz.

Orthogonalization kapsamında yukarıdaki iki başlık için bağımsız toollarımız var 1.si için Bigger Network, Better Optimization (Longer training or Better algorithm) işe yararken, 2.si için regularization or Bigger Dataset işe yarar.

Yani bu hafta kapsamında gördüğümüz modeli iyileştirme sürecini özetlememiz gerekirse:

Reducing (avoidable) bias and variance



- İlk olarak HLP ile TError farkına bakarak avoidable bias ile ilgili fikir ediniriz bu bize, training set üzerinde performansımızı ne kadar daha artırabiliriz ile ilgili bilgi verir. Burada HLP Bayes error'un bir estimate'ı yani bu set üzerinde yapılabilecek minimum olası hatayı temsil eder ki genelde bu HLP bu degree çok uzak değildir.
- İkinci olarak TError ile DError farkına bakarak variance ile ilgili bilgi ediniriz, bu da bize training set performansımızın dev set üzerinde ne kadar işe genelleştirilebildiğini gösterir.
- Bu iki ölçüm sonucuna bakarak hangisine odaklanmamız gerektiğine karar veririz.
- Eğer avoidable bias'ı azaltmaya çalışıyorsak sağ üstteki knob'ları kullanırız.
- Eğer problemin variance olduğunu düşünüyorsak ise, yukarıda belirtilen diğer 3 knob ile modeli iyileştirmeye çalışırız.