W10 - Grad Des. with Large Datasets

In the next few courses we'll talk about large scale machine learning. That is algorithms dealing with big data sets.

Machine Learning & Data

Why do we want to use such large data sets? We've already seen that one of the best ways to get a high performance machine learning system, is if you take a low-bias learning algorithm, and train that on a lot of data.

There is a saying "it's not who has the best algorithm that wins It's who has the most data".

Learning with Large Data Sets

Learning with large data sets comes with it's own unique problems (specifically computational problems)

Let's say our training set size $m = 100\,000\,000$ this is realistic for modern applications.

Let's say we wanna apply linear reg. or log. reg. model and the grad. des. rule will be:

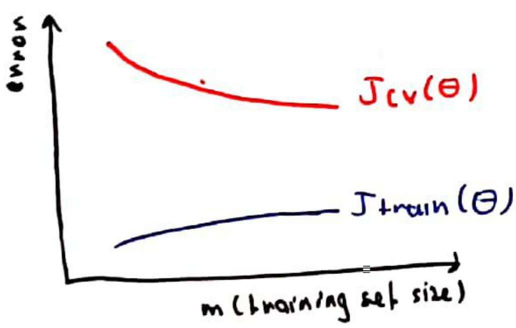
$$\theta_J := \theta_J - \alpha \frac{1}{m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)}) . x_J^{(i)}$$

: Even for a single grad. des. step we need to compute the error for 100 000 000 terms.

Here is a computational problem. We'll talk about how to replace this algorithm or find more efficient ways to compute this derivative.

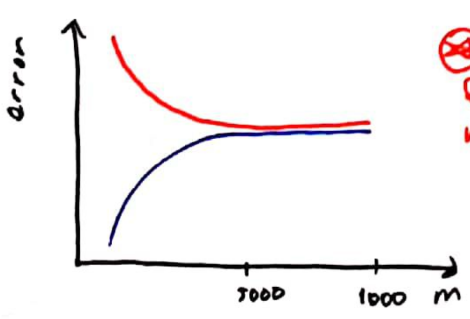
Tabi 100 000 000 examples ile model eğitmeden önce örneğin m=1000 için bir model eğitmek mantıklı. 1000 örnek için learning curves çizilip algoritmanın neye ihtiyacı olduğu belirlenebilir:

If we were to plot the learning curves and if your training obj. looks like



This looks like a high variance learning algorithm and we will be more confident that adding extra training examples would improve performance.

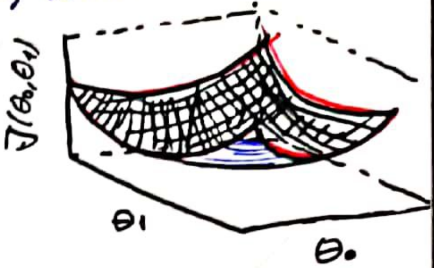
Whereas in contrast if we were to plot the learning curves look like the 2nd figure. Then it looks like the classical high-bias learning algorithm. Durum böyle ise m=1000'den m=100 000 000'e artırmak bir işe yaramaz m=1000'de kalmak daha mantıklı.



Bu durumda yapılabilecek şeylerden biri: adding extra features or adding extra hidden units to the neural network and so on. So that we end up with a situation closer to that on the first figure then this gives us more confidence that trying to add infrastructure to change the algorithm to use much more than 1000 examples that migh be a good use of our time.

W10 - Stochastic Gradient Descent -

→ When we have a very large training set gradient descent becomes computationally very expensive procedure. In this section we'll talk about a modification to the basic gradient descent algorithm called Stochastic Gradient Descent, which will allow us to scale these algorithms to much bigger training sets.



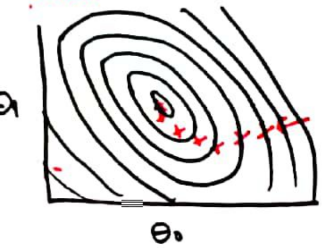
Linear Regression with Gradient Descent

Quick recap: • $\boxed{h(x) = \sum_{j=0}^{n} \theta_j x_j}$ • $\boxed{J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2}$

$$\boxed{\begin{aligned}&\text{Repeat } \{\\ &\quad \theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})\cdot x_j^{(i)}\\ &\quad (\text{for every } j = 0, \dots, n)\\ &\}\end{aligned}}$$

(★) Bu dersin kalanında örnek olarak linear reg. kullanılacak ama Stochastic G.D. is fully general and also applies to other learning alg.s like log. reg., NNs and others based on training grad. des. on a specific training set.

What Grad. Des. Does:



→ Different iterations will take the parameters to the global minimum. The trajectory heads pretty directly to the global minimum.

→ Dediğimiz gibi GD problemi her step için tüm training set'i gezip hesaplama yapması bu yüzden GD'nin diğer bir adı da BATCH GRAD. DES. ↳ batch of the all the training ex.s

(★) We are going to come up with a different alg. that doesn't need to look at all the training examples in every single iteration of θ. But it needs to look at only a single training example in one iteration.

Batch Gradient Descent

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2$$

Repeat {

$$\theta_j := \theta_j - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})\cdot x_j^{(i)}}_{\frac{\partial}{\partial\theta_j}J(\theta)}$$

(for every $j = 0, \dots, n$)

}

Stochastic Gradient Descent

$$\text{cost}(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h(x^{(i)}) - y^{(i)})^2$$

measures how well the hyp. does on a single ex.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{cost}(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset (m training examples)
2. Repeat {

for $i = 1, \dots, m$ {

$$\theta_j := \theta_j - \alpha \underbrace{(h(x^{(i)}) - y^{(i)})\cdot x_j^{(i)}}_{\frac{\partial}{\partial\theta_j}\text{cost}(\theta, (x^{(i)}, y^{(i)}))}$$

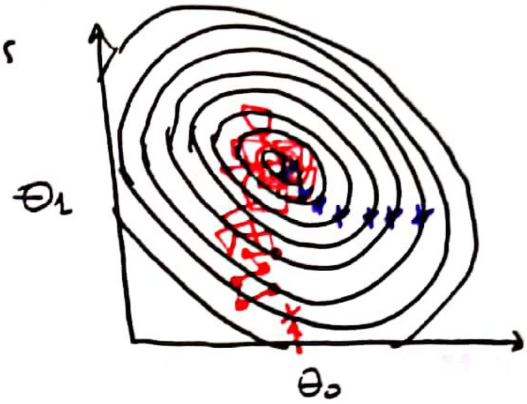
(for $j = 0, \dots, n$)

}

(⊗) What Stochastic Gradient is doing is: It is scanning through the examples. Mesela önce ilk ex. bakar: $(x^{(1)}, y^{(1)})$ diyelim. ve sadece bunun hatasına göre θ'yı update eder. Yani bir θ update'i için m ex. yerine yalnızca 1 ex. kullanır. Tabi $i = 1$ için yalnızca θ'yı update edince yalnızca $x^{(1)}, y^{(1)}$ için performans artışı sağlamayı hedefliyor.

Stochastic Gradient Descent

- Batch GD
- Stochastic GD



1. Randomly shuffle (reorder) training examples
2. Repeat { 1-10 x

    for $i = 1, \ldots, m$ {

    $$\theta_j := \theta_j - \alpha \left(h(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

    (for every $j = 0, \ldots, n$)

    }

}

(*) Batch GD was tend to take a reasonably straight line trajectory to get to the global minimum. In contrast with Stochastic GD every iteration is going to be much faster because we don't need to sum up over all the training ex.s. But every iteration is just trying to fit single training example better.

(*) Stochastic GD will ~~lead~~ generally lead the parameters to move in the direction of the global minimum, but not always. Global min.'e doğru böyle random-looking bir path çizer. As we run the SGD it doesn't actually converge in the same sense as Batch GD does it ends up wandering around continuously in some region that's in some region close to the global minimum.

→ But in practice not converging is not actually a problem, as long as the parameters end up in some region pretty close to the global min the hypothesis will be pretty good. Usually running SGD we get parameters near the global min.

(*) Son olarak dıştaki loop'un ne kadar döneceğine nasıl karar veririz? m boyutuna göre belki doing a single loop may be enough. 1-10 times typical. Eğer m çok büyükse (3 000 000 gibi) 1 loop yetebilir. Denemek lazım.

W10 - Mini-Batch Gradient Descent

→ There is another variation of the GD algorithm. It is called Mini-Batch GD and it can work sometimes a bit faster than SGD.

Mini-Batch GD

→ Batch GD: Use all m examples in each iteration.
→ Stochastic GD: Use 1 example in each iteration
→ Mini-Batch GD: Use b examples in each iteration. b = mini-batch size.

(*) So the MBGD is somewhat in-between BGD and SGD. It is just like Batch GD except that it is using a much smaller batch size

→ Typical choice for the value of b mighbe: b = 2 - 100 generally b = 10

→ we are gonna get b = 10 examples : $(x^{(i)}, y^{(i)}), \dots, (x^{(i+8)}, y^{(i+8)}), \dots (x^{(i+9)}, y^{(i+9)})$ Then perform GD update using this 10 ex.s: $\Theta_J := \Theta_J - \alpha \frac{1}{10} \sum_{k=i}^{i+9} h(x^{(k)}) - y^{(k)} . x_J^{(k)}$ then we will go on the next ten examples for the next iteration.

Mini-Batch GD Algorithm

Say b = 10, m = 1000

Repeat {
  for i = 1, 11, 21, 31, ..., 991 {

$$\Theta_J := \Theta_J - \alpha \frac{1}{10} \sum_{k=i}^{i+9} \left(h(x^{(k)}) - y^{(k)} \right) . x_J^{(k)}$$

  (for every J = 0, ..., n)
  }
}

(*) Compared to Batch GD Mini-Batch GD also allows us to make progress much faster.

(*) How about MBGD vs Sthocastic GD ?

(*) The answer is vectorization. Mini-batch GD is likely to outperform Stochastic GD only if we have a good vectorized implementation.

→ The sum over 10 examples can be performed in a more vectorized way which will allow us to partially parallize our computation over the ten examples. In other words by using appropriate vectorization to compute the rest of the terms we can sometimes partially use the good numerical algebra libraries and parallelize our gradient descent computations over b examples.

(*) One disadvantage of Mini-batch GD is that there is now this extra parameter b which we may have to fiddle with.

w10 - Stochastic GD Convergence -

→ When we are running the SGD how do we make sure that it's converging okay, and how do we tune the learning rate α with SGD? In this part we will see some techniques about these issues:

Checking for Convergence

→ Batch GD: için convergence'tan emin olmak için $J(\Theta)$ - iterations eğrisine bakıyorduk. Eğer $J(\Theta)$ sürekli azalıyorsa GD doğru çalışıyor derdik.

- $\boxed{J(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h(x^{(i)}) - y^{(i)}\right)^2}$ idi.
- m küçük iken iteration vs $J(\Theta)$ grafiği çizdirmek feasable ama m = 3000000 gibi büyük bir training set var ise algoritmayı sürekli pause edip durdurmuş gibi olacağız.

→ Stochastic GD: için convergence'ı anlayabilmek amacıyla ne yapabiliriz?

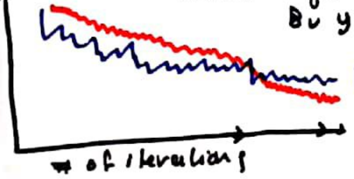
- $\boxed{cost(\Theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}\left(h(x^{(i)}) - y^{(i)}\right)^2}$

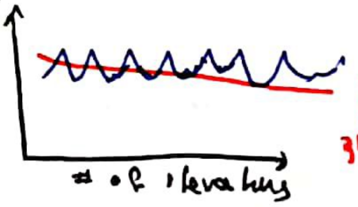
While SGD is learning, compute $cost(\Theta, (x^{(i)}, y^{(i)}))$ before updating Θ using $(x^{(i)}, y^{(i)})$.

Every 1000 iterations (say), plot $cost(\Theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm. Her 1000 iterasyonda bir hesaplanan son 1000 cost'un average'ını plot ettiriyoruz. Bu plot'a bakarak SGD'nin converge edip etmediğini (~~ediyor~~ etmekte olduğunu) anlayabiliriz.

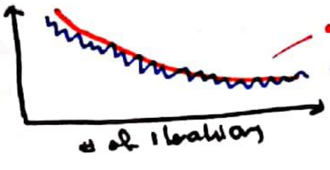
→ Here are some plots of $cost(\Theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 ex.s:

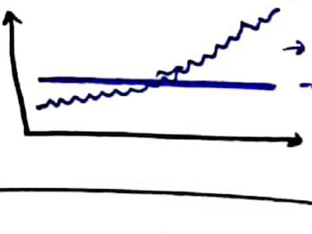
- If we use smaller α: we may get better (but slow) converge. α küçük olduğu için küçük adımlarla gelir ama en sonunda daha küçük bir çapta global min etrafında gezinir. Bu yüzden daha iyi sonuç verebilir.





: Sometimes we see a plot like this
: avg. over 5000 ex.s. için bakınca aslında converge ettiğini görebiliriz. Noise'dan kurtulduk.





→ should use a smaller α!
→ change α or features.

Issue of the Learning Rate

→ α genelde sabit tutulur ve daha önce gördüğümüz gibi glob. min'e doğru ~~bir~~ zikzaklı bir converge elde ederiz. En sonlarda ise global min. etrafında döner durur.

→ Glob. min.'e daha iyi bir convergence için α'yı giderek düşürebiliriz.

E.g. $\alpha = \frac{const1}{iterationNumber + const2}$: α zamanla küçülecek ve glob. min. etrafında daha küçük daireler çizer.

- Bu method çok popüler değil çünkü 2 yeni parametre (const1, ...

W10 - Online Learning -

→ The online learning setting allows us to model problems where we have a continuous flood or continuous stream of data coming and we would like an algorithm to learn from that.

→ Sürekli akan kullanıcı datasından user preferences gibi bilgileri öğrenerek bunu site içinde kullanabiliriz.

Online Learning

Suppose we are a shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service ($y=1$), sometimes not ($y=0$).

→ We want a learning alg. to help us to optimize the asking price.

→ More specifically let's say we come up with features x capture properties of user, of origin/destination and asking price. We want to learn $p(y=1 \mid x; \theta)$ to optimize price.

→ Yani zaten satış yapan bir site yapılan satışlar ve kaçan müşteriler için x (nereden nereye, fiyat teklifi vb...) tutuluyor ve öyle bir θ fit etmek istiyoruz ki her hangi bir örnek için satışın gerçekleşme olasılığını versin. Bu olasılık bilgisini fiyatı değiştirmek için kullanırız.

⊛ NN veya başka bir alg. kullanılabilir. Let's say we will use Logistic Regression.

→ Here is what an online alg. would do:

Repeat forever {
  Get (x,y) corresponding to user. (x: origin-dest.- offered price) (y: 0 or 1)
  Update θ using just (x,y):
  $$\theta_j := \theta_j - \alpha\,(h(x)-y)\cdot x_j \quad (j=0,\dots,n)$$
}

→ We look at only 1 ex. at a time and we don't use it again. Kullanıcı sayısı azsa datayı kaydedip daha sonra bir ağ eğitmek daha mantıklı ama sürekli bir akış varsa bu yöntem iyi çalışır. Bu yöntem ~~ay~~ değişimlere de iyi adapte olur.

-w10 Online Learning II -

Other Online Learning Example

→ Product Search: Using learning alg. to give good search listings to a user.

→ Diyelim ki mobile phone satan bir online store sahibiyiz. User arama çubuğuna "Android phone 1080p camera" gibi şeyler yazıyor biz de 100 telefondan 10 tanesini seçip kullanıcıya sunacağız.

→ Nasıl bir learning algorithm kullanmalı?

- x = features of phone, how many words in user query match name of phone, how many words in each query match description of phone etc.
- Yani her kullanıcı sorgusu için 10 telefon çıkıyorsa her telefon için yani toplamda 10 tane (x, y) olacak. x içinde telefon özellikleri-nin yanı sıra arama sorgusu ile uyuşma özellikleri tutulacak.
- y=1 if user clicks on link. y=0 otherwise. Learn $p(y=1 \mid x; \theta)$
- Bazı linkler x'lere göre daha tıklanır olacak biz de bu linkleri log. reg. ile ayarabiliriz.
- Show the users phone that they have high probability to click on it.

→ Bu probleme predicted Click Through Rate denir predicted CTR

→ Her user aramasında 10 link çıkacağı için 10 (x, y) ex.s olacak bunları θ'yı iyileştirmek için kullanırız daha sonra bu 10 datayı çöpe atabiliriz.

→ Bu kısımdaki alg. leri daha önce gördüğümüz gibi fixed training set ile de formulize edebilirdik. Bu yöntemle continuous learning sağlıyoruz.

- W10 - Map Reduce -

Map Reduce is another approach in large scale ML.

- Let's say we want to fit linear reg. or log. reg. model on some such and say our Batch GD learning rule is:

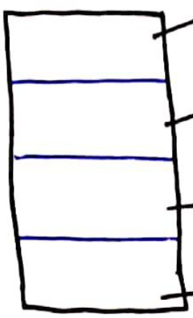
$$\text{Batch GD}: \Theta_J := \Theta_J - \alpha \frac{1}{400} \sum_{i=1}^{400} h(x^{(i)} - y^{(i)}) \cdot x_J^{(i)} \quad \text{say } m = 400$$

- If m is large as we now this is computationally very expensive

What Map Reduce Does:

- splits computational load into different computers

Examples



Machine 1: Use $(x^{(1)}, y^{(1)})$ to $(x^{(100)}, y^{(100)})$

$temp_J^{(1)} = \sum_{i=1}^{100} (h(x^{(i)}) - y^{(i)}) \cdot x_J^{(i)}$

Machine 2: $temp_J^{(2)}$

Machine 3: $temp_J^{(3)}$

Machine 4: Use $x^{(301)}, y^{(301)}$ to $(x^{(400)}, y^{(400)})$

$temp_J^{(4)}$

Send $temp_J$'s to a centeralized master server and combine these results

$\Theta_J := \Theta_J - \alpha \frac{1}{400} (temp_J^{(1)} + temp_J^{(2)} + \dots temp_J^{(4)})$

- In short instead of a single machine to do the all job we can split the training set and divide the computational load.
- Network latency, combining results vb. zaman alsa dahi yukarıdaki şekilde $\simeq 4x$ hızlanma sağlanabilir.
- Map Reduce mantığını tek bilgisayar içinde farklı coreler için de kullanabiliriz.

- Many learning algs can be expressed as computing sums of functions over the training set.

- In order to apply Map-Reduce method to train a NN on 10 machines:

Compute Forward Prop. and Back Prop. on 1/10 of the data to compute the derivative w.r.t. that 1/10 of the data.