

WEEK 10

Large Scale Machine Learning

Gradient Descent with Large Datasets

Learning With Large Datasets

In the next few videos, we'll talk about large scale machine learning. That is, algorithms but viewing with big data sets. If you look back at a recent 5 or 10-year history of machine learning. One of the reasons that learning algorithms work so much better now than even say, 5-years ago, is just the sheer amount of data that we have now and that we can train our algorithms on. In these

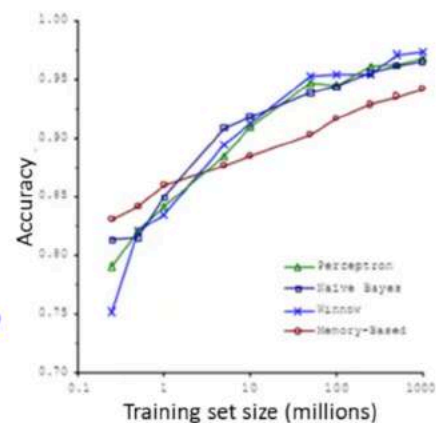
next few videos, we'll talk about algorithms for dealing when we have such massive data sets.

So 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. And so, one early example we have already seen was this example of classifying between confusable words. So, for breakfast, I ate two (TWO) eggs and we saw in this example, these sorts of results, where, you know, so long as you feed the algorithm a lot of data, it seems to do very well. And so it's results like these that has led to the saying in machine learning that often it's not who has the best algorithm that wins. It's who has the most data. So you want to learn from large data sets, at least when we can get such large data sets.

Machine learning and data

Classify between confusable words.
E.g., {to, two, too}, {then, than}.

For breakfast I ate two eggs.

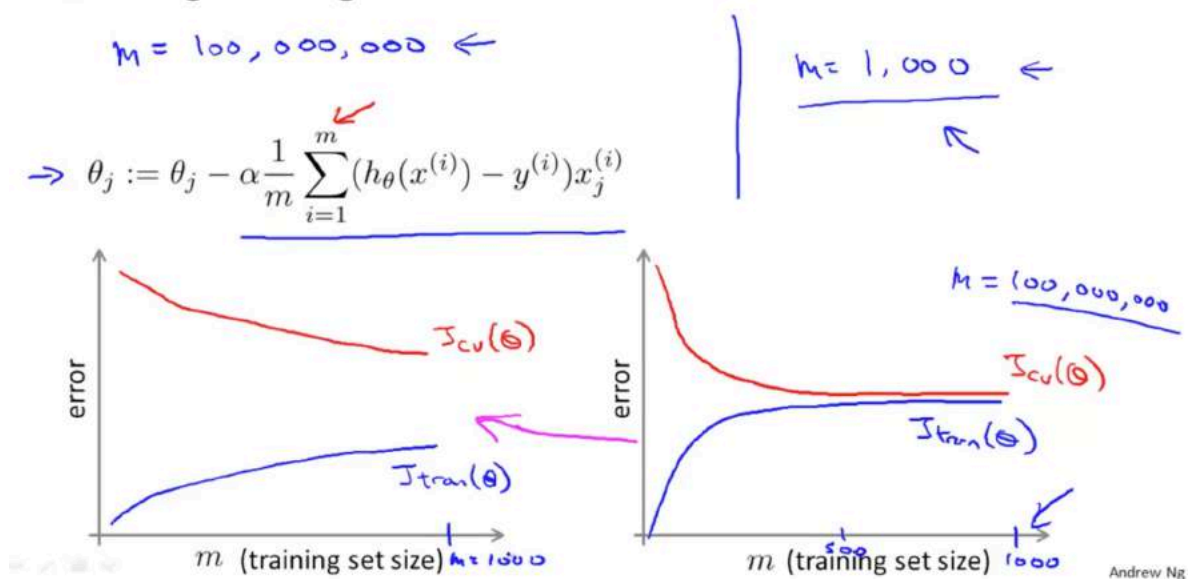


→ “It’s not who has the best algorithm that wins.
It’s who has the most data.”

But learning with large data sets comes with its own unique problems, specifically, computational problems. Let's say your training set size is M equals 100,000,000. And this is actually pretty realistic for many modern data sets. If you look at the US Census data set, if there are, you know, 300 million people in the US, you can usually get hundreds of millions of records. If you look at the amount of traffic that popular websites get, you easily get training sets that are much larger than hundreds of millions of examples. And let's say you want to train a linear regression model, or maybe a logistic regression model, in which case this is the gradient descent rule. And if you look at what you need to do to compute the gradient, which is this term over here, then when M is a hundred million, you need to carry out a summation over a hundred million terms, in order to compute these derivatives terms and to perform a single step of decent. Because of the computational expense of summing over a hundred million entries in order to compute just one step of

gradient descent, in the next few videos we've spoken about techniques for either replacing this with something else or to find more efficient ways to compute this derivative. By the end of this sequence of videos on large scale machine learning, you know how to fit models, linear regression, logistic regression, neural networks and so on even today's data sets with, say, a hundred million examples. Of course, before we put in the effort into training a model with a hundred million examples, We should also ask ourselves, well, why not use just a thousand examples. Maybe we can randomly pick the subsets of a thousand examples out of a hundred million examples and train our algorithm on just a thousand examples. So before investing the effort into actually developing and the software needed to train these massive models is often a good sanity check, if training on just a thousand examples might do just as well. The way to sanity check of using a much smaller training set might do just as well, that is if using a much smaller n equals 1000 size training set, that might do just as well, it is the usual method of plotting the learning curves, so if you were to plot the learning curves and if your training objective were to look like this, that's $J_{\text{train}}(\theta)$. And if your cross-validation set objective, $J_{\text{cv}}(\theta)$ would look like this, then 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 you were to plot the learning curves, if your training objective were to look like this, and if your cross-validation objective were to look like that, then this looks like the classical high-bias learning algorithm. And in the latter case, you know, if you were to plot this up to, say, m equals 1000 and so that is m equals 500 up to m equals 1000, then it seems unlikely that increasing m to a hundred million will do much better and then you'd be just fine sticking to n equals 1000, rather than investing a lot of effort to figure out how the scale of the algorithm. Of course, if you were in the situation shown by the figure on the right, then one natural thing to do would be to add extra features, or add extra hidden units to your neural network and so on, so that you end up with a situation closer to that on the left, where maybe this is up to n equals 1000, and this then gives you more confidence that trying to add infrastructure to change the algorithm to use much more than a thousand examples that might actually be a good use of your time.

Learning with large datasets



Question

Suppose you are facing a supervised learning problem and have a very large dataset ($m = 100,000,000$). How can you tell if using all of the data is likely to perform much better than using a small subset of the data (say $m = 1,000$)?

- ☐ There is no need to verify this; using a larger dataset always gives much better performance.
- ☐ Plot $J_{\text{train}}(\theta)$ as a function of the number of iterations of the optimization algorithm (such as gradient descent).
- ☐ Plot a learning curve ($J_{\text{train}}(\theta)$ and $J_{\text{CV}}(\theta)$, plotted as a function of m) for some range of values of m (say up to $m = 1,000$) and verify that the algorithm has bias when m is small.
- ☒ Plot a learning curve for a range of values of m and verify that the algorithm has high variance when m is small.

Correct

So in large-scale machine learning, we like to come up with computationally reasonable ways, or computationally efficient ways, to deal with very big data sets. In the next few videos, we'll see two main ideas. The first is called stochastic gradient descent and the second is called Map Reduce, for viewing with very big data sets. And after you've learned about these methods, hopefully that will allow you to scale up your learning algorithms to big data and allow you to get much better performance on many different applications.

Stochastic Gradient Descent

For many learning algorithms, among them linear regression, logistic regression and neural networks, the way we derive the algorithm was by coming up with a cost function or coming up with an optimization objective. And then using an algorithm like gradient descent to minimize that cost function. We have a very large training set gradient descent becomes a computationally very expensive procedure. In this video, 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.

Suppose you are training a linear regression model using gradient descent. As a quick recap, the hypothesis will look like this, and the cost function will look like this, which is the sum of one half of the average square error of your hypothesis on your m training examples, and the cost function we've already seen looks like this sort of bow-shaped function. So, plotted as function of the parameters θ_0 and θ_1 , the cost function J is a sort of a bow-shaped function. And gradient descent looks like this, where in the inner loop of gradient descent you repeatedly update the parameters θ using that expression. Now in the rest of this video, I'm going to keep using linear regression as the running example. But the ideas here, the ideas of Stochastic gradient descent is fully general and also applies to other learning algorithms like logistic regression, neural networks and other algorithms that are based on training gradient descent on a specific training set.

Linear regression with gradient descent

$$\rightarrow h_{\theta}(x) = \sum_{j=0}^n \theta_j x_j$$

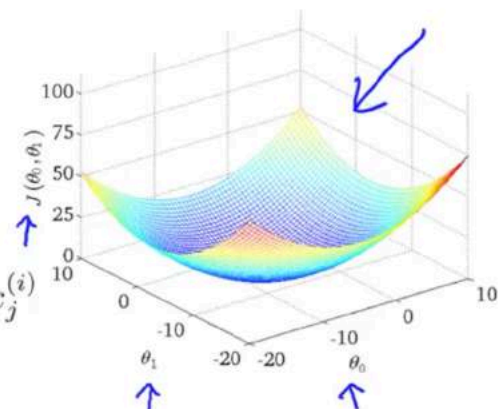
$$\rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Repeat {

$$\rightarrow \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

}



So here's a picture of what gradient descent does, if the parameters are initialized to the point there then as you run gradient descent different iterations of gradient descent will take the parameters to the global minimum.

So take a trajectory that looks like that and heads pretty directly to the global minimum. Now, the problem with gradient descent is that if m is large. Then computing this derivative term can be very expensive, because the surprise, summing over all m examples. So if m is 300 million, alright. So in the United States, there are about 300 million people. And so the US or United States census data may have on the order of that many records. So you want to fit the linear regression model to that then you need to sum over 300 million records. And that's very expensive. To give the algorithm a name, this particular version of gradient descent is also called Batch gradient descent. And the term Batch refers to the fact that we're looking at all of the training examples at a time. We call it sort of a batch of all of the training examples. And it really isn't the, maybe the best name but this is what machine learning people call this particular version of gradient descent. And if you imagine really that you have 300 million census records stored away on disc. The way this algorithm works is you need to read into your computer memory all 300 million records in order to compute this derivative term. You need to stream all of these records through computer because you can't store all your records in computer memory. So you need to read through them and slowly, you know, accumulate the sum in order to compute the derivative. And then having done all that work, that allows you to take one step of gradient descent. And now you need to do the whole thing again. You know, scan through all 300 million records, accumulate these sums. And having done all that work, you can take another little step using gradient descent. And then do that again. And then you take yet a third step. And so on. And so it's gonna take a long time in order to get the algorithm to converge. In contrast to Batch gradient descent, what we are going to do is come up with a different algorithm that doesn't need to look at all the training examples in every single iteration, but that needs to look at only a single training example in one iteration.

Linear regression with gradient descent

$$h_{\theta}(x) = \sum_{j=0}^n \theta_j x_j$$

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Repeat {

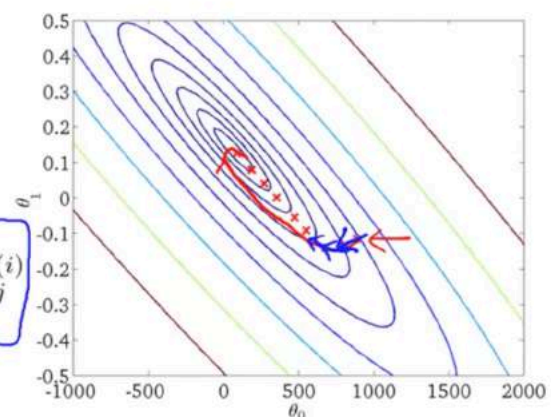
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

}

$m = 300,000,000$

Batch gradient descent



Before moving on to the new algorithm, here's just a Batch gradient descent

algorithm written out again with that being the cost function and that being the update and of course this term here, that's used in the gradient descent rule, that is the partial derivative with respect to the parameters θ_j of our optimization objective, J_{train} of θ . Now, let's look at the more efficient algorithm that scales better to large data sets. In order to work off the algorithms called Stochastic gradient descent, this vectors the cost function in a slightly different way then they define the cost of the parameter θ_j with respect to a training example $x^{(i)}, y^{(i)}$ to be equal to one half times the squared error that my hypothesis incurs on that example, $x^{(i)}, y^{(i)}$. So this cost function term really measures how well is my hypothesis doing on a single example $x^{(i)}, y^{(i)}$. Now you notice that the overall cost function J_{train} can now be written in this equivalent form. So J_{train} is just the average over my m training examples of the cost of my hypothesis on that example $x^{(i)}, y^{(i)}$. Armed with this view of the cost function for linear regression, let me now write out what Stochastic gradient descent does. The first step of Stochastic gradient descent is to randomly shuffle the data set. So by that I just mean randomly shuffle, or randomly reorder your m training examples. It's sort of a standard pre-processing step, come back to this in a minute. But the main work of Stochastic gradient descent is then done in the following. We're going to repeat for i equals 1 through m . So we'll repeatedly scan through my training examples and perform the following update. Gonna update the parameter θ_j as θ_j minus α times $\frac{\partial}{\partial \theta_j} \text{cost}(x^{(i)}, y^{(i)})$. And we're going to do this update as usual for all values of j . Now, you notice that this term over here is exactly what we had inside the summation for Batch gradient descent. In fact, for those of you that are calculus is possible to show that that term here, that's this term here, is equal to the partial derivative with respect to my parameter θ_j of the cost of the parameters θ on $x^{(i)}, y^{(i)}$. Where cost is of course this thing that was defined previously. And just the wrap of the algorithm, let me close my curly braces over there. So what Stochastic gradient descent is doing is it is actually scanning through the training examples. And first it's gonna look at my first training example $x^{(1)}, y^{(1)}$. And then looking at only this first example, it's gonna take like a basically a little gradient descent step with respect to the cost of just this first training example. So in other words, we're going to look at the first example and modify the parameters a little bit to fit just the first training example a little bit better. Having done this inside this inner for-loop is then going to go on to the second training example. And what it's going to do there is take another little step in parameter space, so modify the parameters just a little bit to try to fit just a second training example a little bit better. Having done that, is then going to go onto my third training example. And modify the parameters to try to fit just the third training example a little bit better, and so on until you know, you get through the entire training set. And then this ultra repeat loop may cause it to take multiple passes over the entire training set. This view of Stochastic gradient descent also motivates why we wanted to start by randomly shuffling the data set. This doesn't show us that when we scan through the training site here, that we end up visiting the training examples in some sort of randomly

sorted order. Depending on whether your data already came randomly sorted or whether it came originally sorted in some strange order, in practice this would just speed up the conversions to Stochastic gradient descent just a little bit. So in the interest of safety, it's usually better to randomly shuffle the data set if you aren't sure if it came to you in randomly sorted order. But more importantly another view of Stochastic gradient descent is that it's a lot like descent but rather than wait to sum up these gradient terms over all m training examples, what we're doing is we're taking this gradient term using just one single training example and we're starting to make progress in improving the parameters already. So rather than, you know, waiting 'till taking a path through all 300,000 United States Census records, say, rather than needing to scan through all of the training examples before we can modify the parameters a little bit and make progress towards a global minimum. For Stochastic gradient descent instead we just need to look at a single training example and we're already starting to make progress in this case of parameters towards, moving the parameters towards the global minimum.

Batch gradient descent	Stochastic gradient descent
$\rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$	$\rightarrow cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_{\theta}(x^{(i)}) - y^{(i)})^2$
$J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^m cost(\theta, (x^{(i)}, y^{(i)}))$	
Repeat { $\rightarrow \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$ <div style="text-align: center;"> $\frac{\partial}{\partial \theta_j} J_{train}(\theta)$ (for every $j = 0, \dots, n$) $m = 300,000,000$ </div> }	1. Randomly shuffle dataset. ← 2. Repeat { for $i = 1, \dots, m$ { $\theta_j := \theta_j - \alpha (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$ <div style="text-align: center;"> $\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))$ (for $j = 0, \dots, n$) </div> } } $\rightarrow (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots$

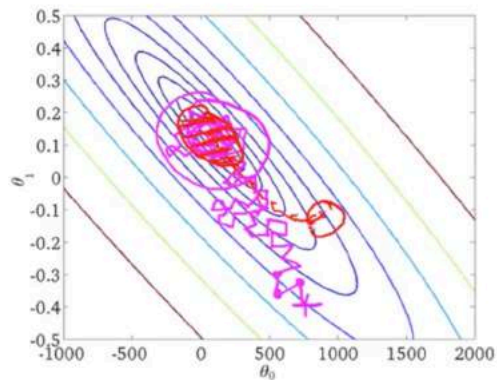
Andrew Ng

So, here's the algorithm written out again where the first step is to randomly shuffle the data and the second step is where the real work is done, where that's the update with respect to a single training example $x^{(i)}, y^{(i)}$. So, let's see what this algorithm does to the parameters. Previously, we saw that when we are using Batch gradient descent, that is the algorithm that looks at all the training examples in time, Batch gradient descent will tend to, you know, take a reasonably straight line trajectory to get to the global minimum like that. In contrast with Stochastic gradient descent every iteration is going to be much faster because we don't need to sum up over all the training examples. But every iteration is just trying to fit single training example better. So, if we were to start stochastic gradient descent, oh, let's start stochastic gradient descent

at a point like that. The first iteration, you know, may take the parameters in that direction and maybe the second iteration looking at just the second example maybe just by chance, we get more unlucky and actually head in a bad direction with the parameters like that. In the third iteration where we tried to modify the parameters to fit just the third training examples better, maybe we'll end up heading in that direction. And then we'll look at the fourth training example and we will do that. The fifth example, sixth example, 7th and so on. And as you run Stochastic gradient descent, what you find is that it will generally move the parameters in the direction of the global minimum, but not always. And so take some more random-looking, circuitous path to reach the global minimum. And in fact as you run Stochastic gradient descent it doesn't actually converge in the same sense as Batch gradient descent does and what it ends up doing is wandering around continuously in some region that's in some region close to the global minimum, but it doesn't just get to the global minimum and stay there. But in practice this isn't a problem because, you know, so long as the parameters end up in some region there maybe it is pretty close to the global minimum. So, as parameters end up pretty close to the global minimum, that will be a pretty good hypothesis and so usually running Stochastic gradient descent we get a parameter near the global minimum and that's good enough for, you know, essentially any, most practical purposes. Just one final detail. In Stochastic gradient descent, we had this outer loop repeat which says to do this inner loop multiple times. So, how many times do we repeat this outer loop? Depending on the size of the training set, doing this loop just a single time may be enough. And up to, you know, maybe 10 times may be typical so we may end up repeating this inner loop anywhere from once to ten times. So if we have a you know, truly massive data set like the this US census gave us that example that I've been talking about with 300 million examples, it is possible that by the time you've taken just a single pass through your training set. So, this is for i equals 1 through 300 million. It's possible that by the time you've taken a single pass through your data set you might already have a perfectly good hypothesis. In which case, you know, this inner loop you might need to do only once if m is very, very large. But in general taking anywhere from 1 through 10 passes through your data set, you know, maybe fairly common. But really it depends on the size of your training set. And if you contrast this to Batch gradient descent. With Batch gradient descent, after taking a pass through your entire training set, you would have taken just one single gradient descent steps. So one of these little baby steps of gradient descent where you just take one small gradient descent step and this is why Stochastic gradient descent can be much faster.

Stochastic gradient descent

- 1. Randomly shuffle (reorder) training examples
 - 2. Repeat { 1-10x
 - for $i := 1, \dots, m$ {
 - $\theta_j := \theta_j - \alpha(h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$
(for every $j = 0, \dots, n$)
- $m = 300,000,000$



Question

Which of the following statements about stochastic gradient descent are true? Check all that apply.

- ☒ When the training set size m is very large, stochastic gradient descent can be much faster than gradient descent.
- ☒ The cost function $J_{\text{train}}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$ should go down with every iteration of batch gradient descent (assuming a well-tuned learning rate α) but not necessarily with stochastic gradient descent.
- ☐ Stochastic gradient descent is applicable only to linear regression but not to other models (such as logistic regression or neural networks).
- ☒ Before beginning the main loop of stochastic gradient descent, it is a good idea to "shuffle" your training data into a random order.

So, that was the Stochastic gradient descent algorithm. And if you implement it, hopefully that will allow you to scale up many of your learning algorithms to much bigger data sets and get much more performance that way.

Mini-Batch Gradient Descent

In the previous video, we talked about Stochastic gradient descent, and how that can be much faster than Batch gradient descent. In this video, let's talk about another variation on these ideas is called Mini-batch gradient descent they can work sometimes even a bit faster than stochastic gradient descent. To summarize the algorithms we talked about so far. In Batch gradient descent we will use all m examples in each generation. Whereas in Stochastic gradient descent we will use a single example in each generation. What Mini-batch gradient descent does is somewhere in between. Specifically, with this algorithm we're going to use b examples in each iteration where b is a parameter called the "mini batch size" so the idea is that this is somewhat in-between Batch gradient descent and Stochastic gradient descent. This is just like batch gradient descent, except that I'm going to use a much smaller batch size. A typical choice for the value of b might be b equals 10, let's say, and a typical range really might be anywhere from b equals 2 up to b equals 100. So that will be a pretty typical range of values for the Mini-batch size. And the idea is that rather than using one example at a time or m examples at a time we will use b examples at a time. So let me just write this out informally, we're going to get, let's say, b . For this example, let's say b equals 10. So we're going to get, the next 10 examples from my training set so that may be some set of examples x_i, y_i . If it's 10 examples then the indexing will be up to $x_{(i+9)}, y_{(i+9)}$ so that's 10 examples altogether and then we'll perform essentially a gradient descent update using these 10 examples. So, that's any rate times one tenth times sum over k equals i through $i+9$ of $h_{\text{subscript } \theta}$ of $x^{(k)}$ minus $y^{(k)}$ times $x^{(k)}$. And so in this expression, where summing the gradient terms over my ten examples. So, that's number ten, that's, you know, my mini batch size and just $i+9$ again, the 9 comes from the choice of the parameter b , and then after this we will then increase, you know, i by tenth, we will go on to the next ten examples and then keep moving like this.

Mini-batch gradient descent

→ Batch gradient descent: Use all m examples in each iteration

→ Stochastic gradient descent: Use 1 example in each iteration

Mini-batch gradient descent: Use b examples in each iteration

$b = \text{mini-batch size. } b = 10. \quad \frac{2-100}{10}$

Get $b = 10$ examples $(x^{(i)}, y^{(i)}), \dots, (x^{(i+9)}, y^{(i+9)})$

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_{\theta}(x^{(k)}) - y^{(k)}) \cdot x_j^{(k)}$$

$i := i + 10$

So just to write out the entire algorithm in full. In order to simplify the indexing for this one at the right top, I'm going to assume we have a mini-batch size of ten and a training set size of a thousand, what we're going to do is have this sort of form, for i equals 1 and that in 21's the stepping, in steps of 10 because we look at 10 examples at a time. And then we perform this sort of gradient descent update using ten examples at a time so this 10 and this $i+9$ those are consequence of having chosen my mini-batch to be ten. And you know, this ultimate four-loop, this ends at 991 here because if I have 1000 training samples then I need 100 steps of size 10 in order to get through my training set. So this is mini-batch gradient descent. Compared to batch gradient descent, this also allows us to make progress much faster. So we have again our running example of, you know, U.S. Census data with 300 million training examples, then what we're saying is after looking at just the first 10 examples we can start to make progress in improving the parameters θ so we don't need to scan through the entire training set. We just need to look at the first 10 examples and this will start letting us make progress and then we can look at the second ten examples and modify the parameters a little bit again and so on. So, that is why Mini-batch gradient descent can be faster than batch gradient descent. Namely, you can start making progress in modifying the parameters after looking at just ten examples rather than needing to wait 'till you've scan through every single training example of 300 million of them. So, how about Mini-batch gradient descent versus Stochastic gradient descent. So, why do we want to look at b examples at a time rather than look at just a single example at a time as the Stochastic gradient descent? The answer is in vectorization. In particular, Mini-batch gradient descent is likely to outperform Stochastic gradient descent only if you have a good vectorized implementation. In that case, the sum over 10 examples can be performed in a more vectorized way which will allow you to partially parallelize your computation over the ten examples. So, in other words, by using appropriate vectorization to compute the rest of the terms, you can sometimes partially use the good numerical algebra libraries and parallelize your gradient computations over the b examples, whereas if you were looking at just a single example of time with Stochastic gradient descent then, you know, just looking at one example at a time their isn't much to parallelize over. At least there is less to parallelize over. One disadvantage of Mini-batch gradient descent is that there is now this extra parameter b , the Mini-batch size which you may have to fiddle with, and which may therefore take time. But if you have a good vectorized implementation this can sometimes run even faster that Stochastic gradient descent.

Mini-batch gradient descent

Say $b = 10$, $m = 1000$.

Repeat {

→ for $i = 1, 11, 21, 31, \dots, 991$ {

→ $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_{\theta}(x^{(k)}) - y^{(k)}) x_j^{(k)}$

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

}

}

$$m = \frac{300,000,000}{10}$$

→ b examples

→ 1 example

Vectorization

$$b = \frac{10}{1}$$

Question

Suppose you use mini-batch gradient descent on a training set of size m , and you use a mini-batch size of b . The algorithm becomes the same as batch gradient descent if:

- ☐ $b = 1$
- ☐ $b = m / 2$
- ☒ $b = m$

Correct

- ☐ None of the above

So that was Mini-batch gradient descent which is an algorithm that in some sense does something that's somewhat in between what Stochastic gradient descent does and what Batch gradient descent does. And if you choose their reasonable value of b . I usually use b equals 10, but, you know, other values, anywhere from say 2 to 100, would be reasonably common. So we choose value of b and if you use a good vectorized implementation, sometimes it can be faster than both Stochastic gradient descent and faster than Batch gradient descent.

Stochastic Gradient Descent Convergence

You now know about the stochastic gradient descent algorithm. But when you're running the algorithm, how do you make sure that it's completely debugged and is converging okay? Equally important, how do you tune the learning rate α with Stochastic Gradient Descent. In this video we'll talk about some techniques for doing these things, for making sure it's converging and for picking the learning rate α .

Back when we were using batch gradient descent, our standard way for making sure that gradient descent was converging was we would plot the optimization cost function as a function of the number of iterations. So that was the cost function and we would make sure that this cost function is decreasing on every iteration. When the training set sizes were small, we could do that because we could compute the sum pretty efficiently. But when you have a massive training set size then you don't want to have to pause your algorithm periodically. You don't want to have to pause stochastic gradient descent periodically in order to compute this cost function since it requires a sum of your entire training set size. And the whole point of stochastic gradient was that you wanted to start to make progress after looking at just a single example without needing to occasionally scan through your entire training set right in the middle of the algorithm, just to compute things like the cost function of the entire training set. So for stochastic gradient descent, in order to check the algorithm is converging, here's what we can do instead. Let's take the definition of the cost that we had previously. So the cost of the parameters θ with respect to a single training example is just one half of the square error on that training example. Then, while stochastic gradient descent is learning, right before we train on a specific example. So, in stochastic gradient descent we're going to look at the examples x_i, y_i , in order, and then sort of take a little update with respect to this example. And we go on to the next example, x_{i+1}, y_{i+1} , and so on, right? That's what stochastic gradient descent does. So, while the algorithm is looking at the example x_i, y_i , but before it has updated the parameters θ using that an example, let's compute the cost of that example. Just to say the same thing again, but using slightly different words. A stochastic gradient descent is scanning through our training set right before we have updated θ using a specific training example $x(i)$ comma $y(i)$ let's compute how well our hypothesis is doing on that training example. And we want to do this before updating θ because if we've just updated θ using example, you know, that it might be doing better on that example than what would be representative. Finally, in order to check for the convergence of stochastic gradient descent, what we can do is

every, say, every thousand iterations, we can plot these costs that we've been computing in the previous step. We can plot those costs average over, say, the last thousand examples processed by the algorithm. And if you do this, it kind of gives you a running estimate of how well the algorithm is doing. on, you know, the last 1000 training examples that your algorithm has seen. So, in contrast to computing J_{train} periodically which needed to scan through the entire training set. With this other procedure, well, as part of stochastic gradient descent, it doesn't cost much to compute these costs as well right before updating to parameter theta. And all we're doing is every thousand integrations or so, we just average the last 1,000 costs that we computed and plot that. And by looking at those plots, this will allow us to check if stochastic gradient descent is converging.

Checking for convergence

→ Batch gradient descent:

→ Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent.

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \quad M = 300,000,000$$

→ Stochastic gradient descent:

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

→ During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.

→ $(x^{(i)}, y^{(i)})$, $(x^{(i+1)}, y^{(i+1)})$, ...

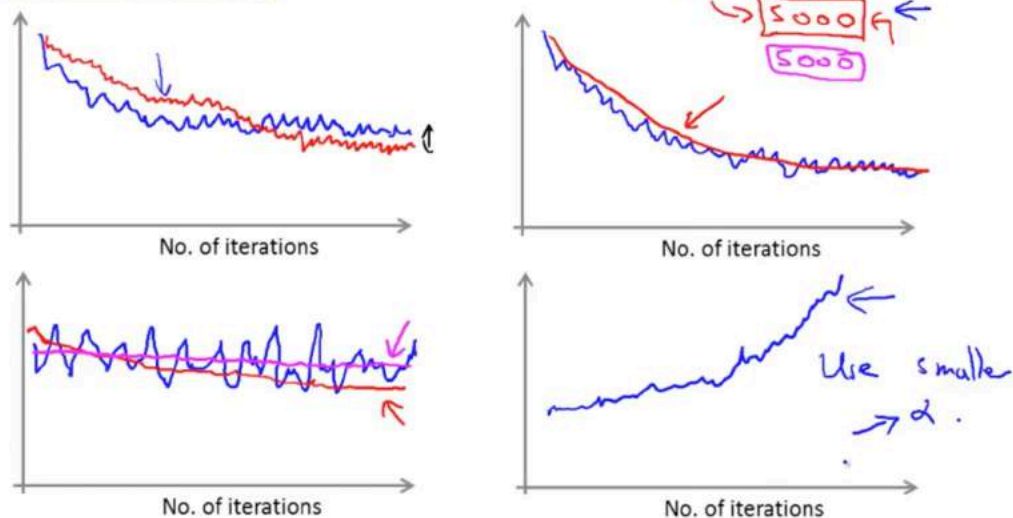
→ Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm.

So here are a few examples of what these plots might look like. Suppose you have plotted the cost average over the last thousand examples, because these are averaged over just a thousand examples, they are going to be a little bit noisy and so, it may not decrease on every single iteration. Then if you get a figure that looks like this, So the plot is noisy because it's average over, you know, just a small subset, say a thousand training examples. If you get a figure that looks like this, you know that would be a pretty decent run with the algorithm, maybe, where it looks like the cost has gone down and then this plateau that looks kind of flattened out, you know, starting from around that point. look like, this is what your cost looks like then maybe your learning algorithm has converged. If you want to try using a smaller learning rate, something you might see is that the algorithm may initially learn more slowly so the cost goes down more slowly. But then eventually you have a smaller learning rate is actually possible for the algorithm to end up at a, maybe very slightly better solution. So the red line may represent the behavior of stochastic gradient descent using a slower, using a smaller leaning rate. And the reason this is the case is because, you remember, stochastic gradient

descent doesn't just converge to the global minimum, is that what it does is the parameters will oscillate a bit around the global minimum. And so by using a smaller learning rate, you'll end up with smaller oscillations. And sometimes this little difference will be negligible and sometimes with a smaller than you can get a slightly better value for the parameters. Here are some other things that might happen. Let's say you run stochastic gradient descent and you average over a thousand examples when plotting these costs. So, you know, here might be the result of another one of these plots. Then again, it kind of looks like it's converged. If you were to take this number, a thousand, and increase to averaging over 5 thousand examples. Then it's possible that you might get a smoother curve that looks more like this. And by averaging over, say 5,000 examples instead of 1,000, you might be able to get a smoother curve like this. And so that's the effect of increasing the number of examples you average over. The disadvantage of making this too big of course is that now you get one data point only every 5,000 examples. And so the feedback you get on how well your learning algorithm is doing is, sort of, maybe it's more delayed because you get one data point on your plot only every 5,000 examples rather than every 1,000 examples. Along a similar vein some times you may run a gradient descent and end up with a plot that looks like this. And with a plot that looks like this, you know, it looks like the cost just is not decreasing at all. It looks like the algorithm is just not learning. It's just, looks like this here a flat curve and the cost is just not decreasing. But again if you were to increase this to averaging over a larger number of examples it is possible that you see something like this red line it looks like the cost actually is decreasing, it's just that the blue line averaging over 2, 3 examples, the blue line was too noisy so you couldn't see the actual trend in the cost actually decreasing and possibly averaging over 5,000 examples instead of 1,000 may help. Of course we averaged over a larger number examples that we've averaged here over 5,000 examples, I'm just using a different color, it is also possible that you that see a learning curve ends up looking like this. That it's still flat even when you average over a larger number of examples. And as you get that, then that's maybe just a more firm verification that unfortunately the algorithm just isn't learning much for whatever reason. And you need to either change the learning rate or change the features or change something else about the algorithm. Finally, one last thing that you might see would be if you were to plot these curves and you see a curve that looks like this, where it actually looks like it's increasing. And if that's the case then this is a sign that the algorithm is diverging. And what you really should do is use a smaller value of the learning rate α . So hopefully this gives you a sense of the range of phenomena you might see when you plot these cost average over some range of examples as well as suggests the sorts of things you might try to do in response to seeing different plots. So if the plots looks too noisy, or if it wiggles up and down too much, then try increasing the number of examples you're averaging over so you can see the overall trend in the plot better. And if you see that the errors are actually increasing, the costs are actually increasing, try using a smaller value of α .

Checking for convergence

Plot $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples



Andre

Finally, it's worth examining the issue of the learning rate just a little bit more. We saw that when we run stochastic gradient descent, the algorithm will start here and sort of meander towards the minimum. And then it won't really converge, and instead it'll wander around the minimum forever. And so you end up with a parameter value that is hopefully close to the global minimum that won't be exact at the global minimum. In most typical implementations of stochastic gradient descent, the learning rate α is typically held constant. And so what you end up with is exactly a picture like this. If you want stochastic gradient descent to actually converge to the global minimum, there's one thing which you can do which is you can slowly decrease the learning rate α over time. So, a pretty typical way of doing that would be to set α equals some constant 1 divided by iteration number plus constant 2. So, iteration number is the number of iterations you've run of stochastic gradient descent, so it's really the number of training examples you've seen. And const 1 and const 2 are additional parameters of the algorithm that you might have to play with a bit in order to get good performance. One of the reasons people tend not to do this is because you end up needing to spend time playing with these 2 extra parameters, constant 1 and constant 2, and so this makes the algorithm more finicky. You know, it's just more parameters able to fiddle with in order to make the algorithm work well. But if you manage to tune the parameters well, then the picture you can get is that the algorithm will actually meander around towards the minimum, but as it gets closer because you're decreasing the learning rate the meanderings will get smaller and smaller until it's pretty much just to the global minimum. I hope this makes sense, right? And the reason this formula makes sense is because as the algorithm runs, the iteration number becomes large. So α will slowly become small, and so you

take smaller and smaller steps until it hopefully converges to the global minimum. So If you do slowly decrease alpha to zero you can end up with a slightly better hypothesis. But because of the extra work needed to fiddle with the constants and because frankly usually we're pretty happy with any parameter value that is, you know, pretty close to the global minimum. Typically this process of decreasing alpha slowly is usually not done and keeping the learning rate alpha constant is the more common application of stochastic gradient descent although you will see people use either version.

Stochastic gradient descent

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

$$J_{\text{train}}(\theta) = \frac{1}{2m} \sum_{i=1}^m \text{cost}(\theta, (x^{(i)}, y^{(i)}))$$

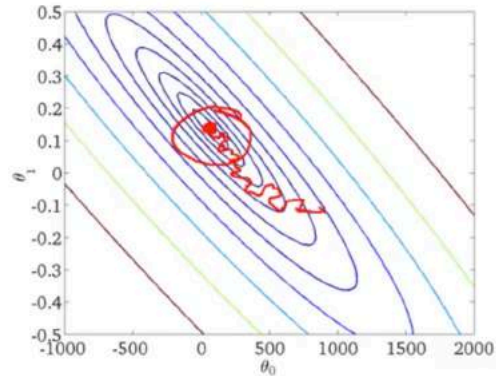
1. Randomly shuffle dataset.
2. Repeat {

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

$\theta_j := \theta_j - \alpha(h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$
 (for $j = 0, \dots, n$)

 }

 }



Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (E.g. $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$) $\alpha \rightarrow 0$

Andrew Ng

Question

Which of the following statements about stochastic gradient descent are true? Check all that apply.

- ☐ Picking a learning rate α that is very small has no disadvantage and can only speed up learning.
- ☒ If we reduce the learning rate α (and run stochastic gradient descent long enough), it's possible that we may find a set of better parameters than with larger α .
- ☐ If we want stochastic gradient descent to converge to a (local) minimum rather than wander of "oscillate" around it, we should slowly increase α over time.
- ☒ If we plot $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$ (averaged over the last 1000 examples) and stochastic gradient descent does not seem to be reducing the cost, one possible problem may be that the learning rate α is poorly tuned.

To summarize in this video we talk about a way for approximately monitoring how the stochastic gradient descent is doing in terms for optimizing the cost function. And this is a method that does not require scanning over the entire training set periodically to compute the cost function on the entire training set. But instead it looks at say only the last thousand examples or so. And you can use this method both to make sure the stochastic gradient descent is okay and is converging or to use it to tune the learning rate α .

Advanced Topics

Online Learning

In this video, I'd like to talk about a new large-scale machine learning setting called the online learning setting. The online learning setting allows us to model problems where we have a continuous flood or a continuous stream of data coming in and we would like an algorithm to learn from that. Today, many of the largest websites, or many of the largest website companies use different versions of online learning algorithms to learn from the flood of users that keep on coming to, back to the website. Specifically, if you have a continuous stream of data generated by a continuous stream of users coming to your website, what you can do is sometimes use an online learning algorithm to learn user preferences from the stream of data and use that to optimize some of the decisions on your website. Suppose you run a shipping service, so, you know, users come and ask you to help ship their package from location A to location B and suppose you run a website, where users repeatedly come and they tell you where they want to send the package from, and where they want to send it to (so the origin and destination) and your website offers to ship the package for some asking price, so I'll ship your package for \$50, I'll ship it for \$20. And based on the price that you offer to the users, the users sometimes chose to use a shipping service; that's a positive example and sometimes they go away and they do not choose to purchase your shipping service. So let's say that we want a learning algorithm to help us to optimize what is the asking price that we want to offer to our users. And specifically, let's say we come up with some sort of features that capture properties of the users. If we know anything about the demographics, they capture, you know, the origin and destination of the package, where they want to ship the package. And what is

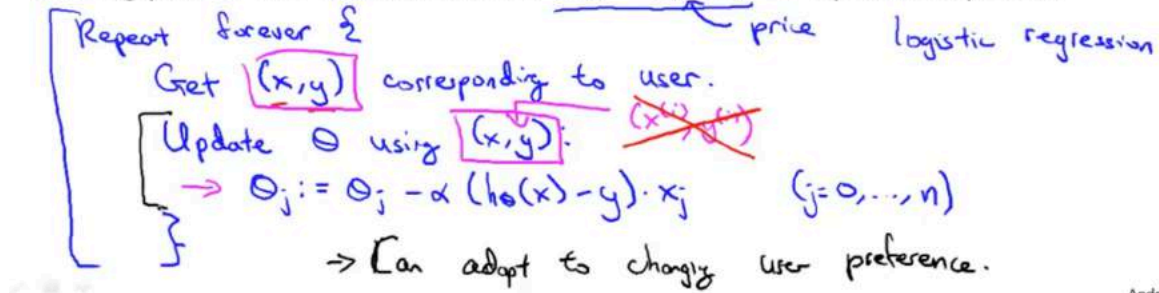
the price that we offer to them for shipping the package. and what we want to do is learn what is the probability that they will elect to ship the package, using our shipping service given these features, and again just as a reminder these features X also captures the price that we're asking for. And so if we could estimate the chance that they'll agree to use our service for any given price, then we can try to pick a price so that they have a pretty high probability of choosing our website while simultaneously hopefully offering us a fair return, offering us a fair profit for shipping their package. So if we can learn this property of y equals 1 given any price and given the other features we could really use this to choose appropriate prices as new users come to us. So in order to model the probability of y equals 1, what we can do is use logistic regression or neural network or some other algorithm like that. But let's start with logistic regression. Now if you have a website that just runs continuously, here's what an online learning algorithm would do. I'm gonna write repeat forever. This just means that our website is going to, you know, keep on staying up. What happens on the website is occasionally a user will come and for the user that comes we'll get some x, y pair corresponding to a customer or to a user on the website. So the features x are, you know, the origin and destination specified by this user and the price that we happened to offer to them this time around, and y is either one or zero depending on whether or not they chose to use our shipping service. Now once we get this $\{x, y\}$ pair, what an online learning algorithm does is then update the parameters θ using just this example x, y , and in particular we would update my parameters θ as θ_j get updated as θ_j minus the learning rate α times my usual gradient descent rule for logistic regression. So we do this for j equals zero up to n , and that's my close curly brace. So, for other learning algorithms instead of writing X, Y , right, I was writing things like X_i, Y_i but in this online learning setting where actually discarding the notion of there being a fixed training set instead we have an algorithm. Now what happens as we get an example and then we learn using that example like so and then we throw that example away. We discard that example and we never use it again and so that's why we just look at one example at a time. We learn from that example. We discard it. Which is why, you know, we're also doing away with this notion of there being this sort of fixed training set indexed by i . And, if you really run a major website where you really have a continuous stream of users coming, then this sort of online learning algorithm is actually a pretty reasonable algorithm. Because of data is essentially free if you have so much data, that data is essentially unlimited then there is really may be no need to look at a training example more than once. Of course if we had only a small number of users then rather than using an online learning algorithm like this, you might be better off saving away all your data in a fixed training set and then running some algorithm over that training set. But if you really have a continuous stream of data, then an online learning algorithm can be very effective. I should mention also that one interesting effect of this sort of online learning algorithm is that it can adapt to changing user preferences. And in particular, if over time because of changes in the economy maybe users start to become more price sensitive and willing

to pay, you know, less willing to pay high prices. Or if they become less price sensitive and they're willing to pay higher prices. Or if different things become more important to users, if you start to have new types of users coming to your website. This sort of online learning algorithm can also adapt to changing user preferences and kind of keep track of what your changing population of users may be willing to pay for. And it does that because if your pool of users changes, then these updates to your parameters θ will just slowly adapt your parameters to whatever your latest pool of users looks like.

Online learning

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 ($y = 0$).

Features x capture properties of user, of origin/destination and asking price. We want to learn $p(y = 1|x; \theta)$ to optimize price.



Andrew

Here's another example of a sort of application to which you might apply online learning. this is an application in product search in which we want to apply learning algorithm to learn to give good search listings to a user. Let's say you run an online store that sells phones - that sells mobile phones or sells cell phones. And you have a user interface where a user can come to your website and type in the query like "Android phone 1080p camera". So 1080p is a type of a specification for a video camera that you might have on a phone, a cell phone, a mobile phone. Suppose, suppose we have a hundred phones in our store. And because of the way our website is laid out, when a user types in a query, if it was a search query, we would like to find a choice of ten different phones to show what to offer to the user. What we'd like to do is have a learning algorithm help us figure out what are the ten phones out of the 100 we should return the user in response to a user-search query like the one here. Here's how we can go about the problem. For each phone and given a specific user query; we can construct a feature vector X . So the feature vector X might capture different properties of the phone. It might capture things like, how similar the user search query is in the phones. We capture things like how many words in the user search query match the name of the phone, how many words in the user search query match the description of the phone and so on.

So the features x capture properties of the phone and it captures things about how similar or how well the phone matches the user query along different dimensions. What we like to do is estimate the probability that a user will click on the link for a specific phone, because we want to show the user phones that they are likely to want to buy, want to show the user phones that they have high probability of clicking on in the web browser. So I'm going to define y equals one if the user clicks on the link for a phone and y equals zero otherwise and what I would like to do is learn the probability the user will click on a specific phone given, you know, the features x , which capture properties of the phone and how well the query matches the phone. To give this problem a name in the language of people that run websites like this, the problem of learning this is actually called the problem of learning the predicted click-through rate, the predicted CTR. It just means learning the probability that the user will click on the specific link that you offer them, so CTR is an abbreviation for click through rate. And if you can estimate the predicted click-through rate for any particular phone, what we can do is use this to show the user the ten phones that are most likely to click on, because out of the hundred phones, we can compute this for each of the 100 phones and just select the 10 phones that the user is most likely to click on, and this will be a pretty reasonable way to decide what ten results to show to the user. Just to be clear, suppose that every time a user does a search, we return ten results what that will do is it will actually give us ten x, y pairs, this actually gives us ten training examples every time a user comes to our website because, because for the ten phone that we chose to show the user, for each of those 10 phones we get a feature vector X , and for each of those 10 phones we show the user we will also get a value for y , we will also observe the value of y , depending on whether or not we clicked on that url or not and so, one way to run a website like this would be to continuously show the user, you know, your ten best guesses for what other phones they might like and so, each time a user comes you would get ten examples, ten x, y pairs, and then use an online learning algorithm to update the parameters using essentially 10 steps of gradient descent on these 10 examples, and then you can throw the data away, and if you really have a continuous stream of users coming to your website, this would be a pretty reasonable way to learn parameters for your algorithm so as to show the ten phones to your users that may be most promising and the most likely to click on. So, this is a product search problem or learning to rank phones, learning to search for phones example. So, I'll quickly mention a few others. One is, if you have a website and you're trying to decide, you know, what special offer to show the user, this is very similar to phones, or if you have a website and you show different users different news articles. So, if you're a news aggregator website, then you can again use a similar system to select, to show to the user, you know, what are the news articles that they are most likely to be interested in and what are the news articles that they are most likely to click on. Closely related to special offers, will we profit from recommendations. And in fact, if you have a collaborative filtering system, you can even imagine a collaborative filtering system giving you additional features to feed into a logistic regression

classifier to try to predict the click through rate for different products that you might recommend to a user. Of course, I should say that any of these problems could also have been formulated as a standard machine learning problem, where you have a fixed training set. Maybe, you can run your website for a few days and then save away a training set, a fixed training set, and run a learning algorithm on that. But these are the actual sorts of problems, where you do see large companies get so much data, that there's really maybe no need to save away a fixed training set, but instead you can use an online learning algorithm to just learn continuously. from the data that users are generating on your website.

Other online learning example:

Product search (learning to search)

User searches for "Android phone 1080p camera" ←

Have 100 phones in store. Will return 10 results.

→ x = features of phone, how many words in user query match name of phone, how many words in query match description of phone, etc.

→ $y = 1$ if user clicks on link. $y = 0$ otherwise.

→ Learn $p(y = 1|x; \theta)$. ← predicted CTR

(x, y) ←
↑ ↑

→ Use to show user the 10 phones they're most likely to click on.

Other examples: Choosing special offers to show user; customized selection of news articles; product recommendation; ...

Question

Some of the advantages of using an online learning algorithm are:

- ☒ It can adapt to changing user tastes (i.e., if $p(y|x; \theta)$ changes over time).

Correct

- ☐ There is no need to pick a learning rate α .

Un-selected is correct

- ☒ It allows us to learn from a continuous stream of data, since we use each example once then no longer need to process it again.

Correct

- ☐ It does not require that good features be chosen for the learning task.

Un-selected is correct

So, that was the online learning setting and as we saw, the algorithm that we apply to it is really very similar to this stochastic gradient descent algorithm, only instead of scanning through a fixed training set, we're instead getting one example from a user, learning from that example, then discarding it and moving on. And if you have a continuous stream of data for some application, this sort of algorithm may be well worth considering for your application. And of course, one advantage of online learning is also that if you have a changing pool of users, or if the things you're trying to predict are slowly changing like your user taste is slowly changing, the online learning algorithm can slowly adapt your learned hypothesis to whatever the latest sets of user behaviors are like as well.

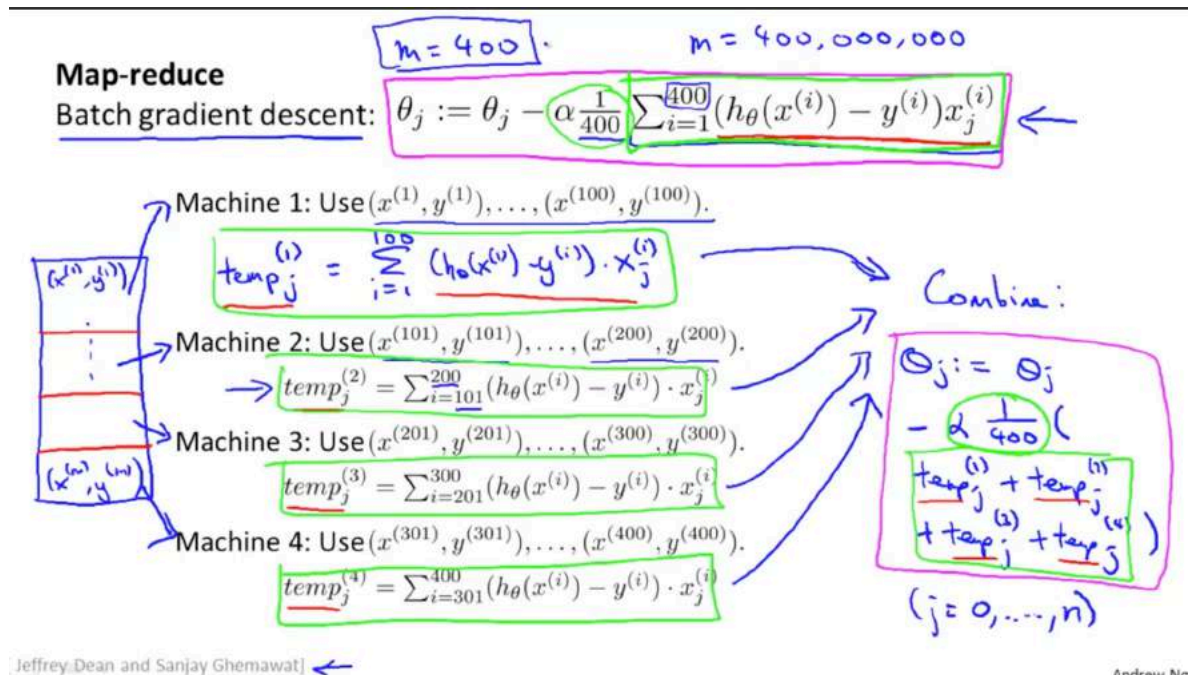
Map Reduce and Data Parallelism

In the last few videos, we talked about stochastic gradient descent, and, you know, other variations of the stochastic gradient descent algorithm, including those adaptations to online learning, but all of those algorithms could be run on one machine, or could be run on one computer. And some machine learning problems are just too big to run on one machine, sometimes maybe you just so much data you just don't ever want to run all that data through a single

computer, no matter what algorithm you would use on that computer. So in this video I'd like to talk about different approach to large scale machine learning, called the map reduce approach. And even though we have quite a few videos on stochastic gradient descent and we're going to spend relative less time on map reduce--don't judge the relative importance of map reduce versus the gradient descent based on the amount amount of time I spend on these ideas in particular. Many people will say that map reduce is at least an equally important, and some would say an even more important idea compared to gradient descent, only it's relatively simpler to explain, which is why I'm going to spend less time on it, but using these ideas you might be able to scale learning algorithms to even far larger problems than is possible using stochastic gradient descent.

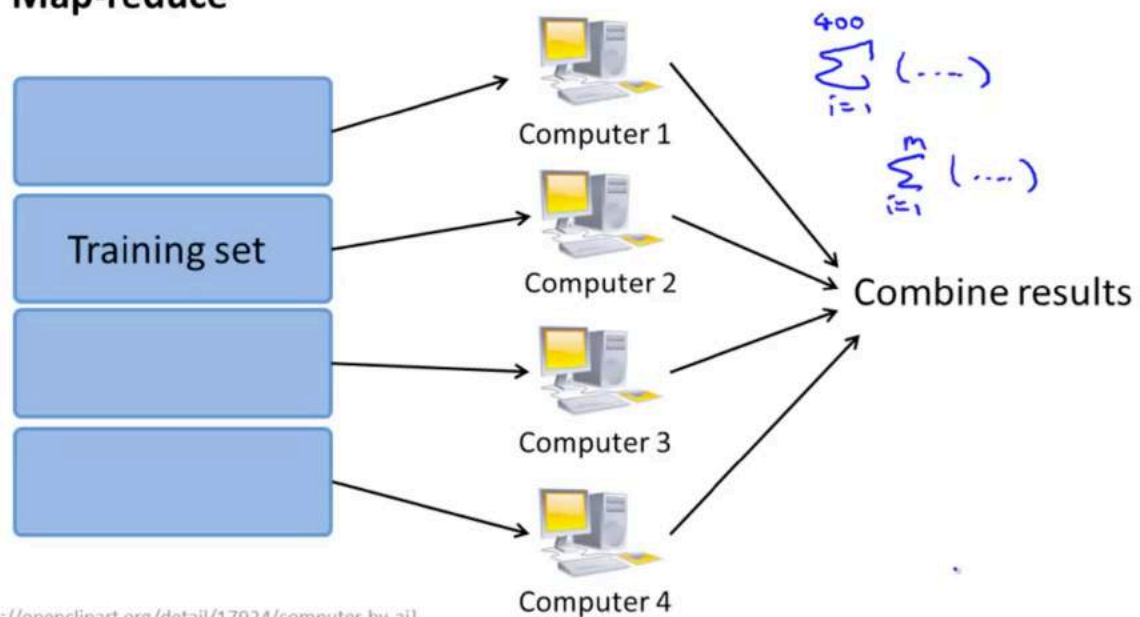
Here's the idea. Let's say we want to fit a linear regression model or a logistic regression model or some such, and let's start again with batch gradient descent, so that's our batch gradient descent learning rule. And to keep the writing on this slide tractable, I'm going to assume throughout that we have m equals 400 examples. Of course, by our standards, in terms of large scale machine learning, you know m might be pretty small and so, this might be more commonly applied to problems, where you have maybe closer to 400 million examples, or some such, but just to make the writing on the slide simpler, I'm going to pretend we have 400 examples. So in that case, the batch gradient descent learning rule has this 400 and the sum from i equals 1 through 400 through my 400 examples here, and if m is large, then this is a computationally expensive step. So, what the MapReduce idea does is the following, and I should say the map reduce idea is due to two researchers, Jeff Dean and Sanjay Gimawat. Jeff Dean, by the way, is one of the most legendary engineers in all of Silicon Valley and he kind of built a large fraction of the architectural infrastructure that all of Google runs on today. But here's the map reduce idea. So, let's say I have some training set, if we want to denote by this box here of X Y pairs, where it's X_1, Y_1 , down to my 400 examples, X_m, Y_m . So, that's my training set with 400 training examples. In the MapReduce idea, one way to do, is split this training set in to different subsets. I'm going to. assume for this example that I have 4 computers, or 4 machines to run in parallel on my training set, which is why I'm splitting this into 4 machines. If you have 10 machines or 100 machines, then you would split your training set into 10 pieces or 100 pieces or what have you. And what the first of my 4 machines is to do, say, is use just the first one quarter of my training set--so use just the first 100 training examples. And in particular, what it's going to do is look at this summation, and compute that summation for just the first 100 training examples. So let me write that up I'm going to compute a variable $temp_1$ the first machine J equals sum from equals 1 through 100, and then I'm going to plug in exactly that term there--so I have $X - \theta$, X_i , minus Y_i times X_{ij} , right? So that's just that gradient descent term up there. And then similarly, I'm going to take the second quarter of my data and send it to my

second machine, and my second machine will use training examples 101 through 200 and you will compute similar variables of a temp to j which is the same sum for index from examples 101 through 200. And similarly machines 3 and 4 will use the third quarter and the fourth quarter of my training set. So now each machine has to sum over 100 instead of over 400 examples and so has to do only a quarter of the work and thus presumably it could do it about four times as fast. Finally, after all these machines have done this work, I am going to take these temp variables and put them back together. So I take these variables and send them all to a You know centralized master server and what the master will do is combine these results together. and in particular, it will update my parameters theta j according to theta j gets updated as theta j minus Of the learning rate alpha times one over 400 times temp, 1, J, plus temp 2j plus temp 3j plus temp 4j and of course we have to do this separately for J equals 0. You know, up to and within this number of features. So operating this equation into I hope it's clear. So what this equation is doing is exactly the same is that when you have a centralized master server that takes the results, the ten one j the ten two j ten three j and ten four j and adds them up and so of course the sum of these four things. Right, that's just the sum of this, plus the sum of this, plus the sum of this, plus the sum of that, and those four things just add up to be equal to this sum that we're originally computing a batch stream descent. And then we have the alpha times 1 of 400, alpha times 1 of 100, and this is exactly equivalent to the batch gradient descent algorithm, only, instead of needing to sum over all four hundred training examples on just one machine, we can instead divide up the work load on four machines.



So, here's what the general picture of the MapReduce technique looks like. We have some training sets, and if we want to parallelize across four machines, we are going to take the training set and split it, you know, equally. Split it as evenly as we can into four subsets. Then we are going to take the 4 subsets of the training data and send them to 4 different computers. And each of the 4 computers can compute a summation over just one quarter of the training set, and then finally take each of the computers takes the results, sends them to a centralized server, which then combines the results together. So, on the previous line in that example, the bulk of the work in gradient descent, was computing the sum from i equals 1 to 400 of something. So more generally, sum from i equals 1 to m of that formula for gradient descent. And now, because each of the four computers can do just a quarter of the work, potentially you can get up to a 4x speed up. In particular, if there were no network latencies and no costs of the network communications to send the data back and forth, you can potentially get up to a 4x speed up. Of course, in practice, because of network latencies, the overhead of combining the results afterwards and other factors, in practice you get slightly less than a 4x speedup. But, none the less, this sort of macro juice approach does offer us a way to process much larger data sets than is possible using a single computer.

Map-reduce



<http://opencitart.org/detail/17924/computer-by-aj>

An

If you are thinking of applying Map Reduce to some learning algorithm, in order to speed this up. By paralleling the computation over different computers, the key question to ask yourself is, can your learning algorithm be expressed as a summation over the training set? And it turns out that many learning algorithms can actually be expressed as computing sums of functions over the training set and the computational expense of running them on large data sets is because they need to sum over a very large training set. So, whenever your

learning algorithm can be expressed as a sum over the training set and whenever the bulk of the work of the learning algorithm can be expressed as the sum over the training set, then map reduces might be a good candidate for scaling your learning algorithms through very, very good data sets. Let's just look at one more example. Let's say that we want to use one of the advanced optimization algorithms. So, things like, you know, LBFGS, conjugate gradient and so on, and let's say we want to train a logistic regression of the algorithm. For that, we need to compute two main quantities. One is for the advanced optimization algorithms like, you know, LBFGS and conjugate gradient. We need to provide it a routine to compute the cost function of the optimization objective. And so for logistic regression, you remember that a cost function has this sort of sum over the training set, and so if you're parallelizing over ten machines, you would split up the training set onto ten machines and have each of the ten machines compute the sum of this quantity over just one tenth of the training data. Then, the other thing that the advanced optimization algorithms need, is a routine to compute these partial derivative terms. Once again, these derivative terms, for which it's a logistic regression, can be expressed as a sum over the training set, and so once again, similar to our earlier example, you would have each machine compute that summation over just some small fraction of your training data. And finally, having computed all of these things, they could then send their results to a centralized server, which can then add up the partial sums. This corresponds to adding up those $\frac{1}{10}$ or $\frac{1}{10}$ variables, which were computed locally on machine number i , and so the centralized server can sum these things up and get the overall cost function and get the overall partial derivative, which you can then pass through the advanced optimization algorithm. So, more broadly, by taking other learning algorithms and expressing them in sort of summation form or by expressing them in terms of computing sums of functions over the training set, you can use the MapReduce technique to parallelize other learning algorithms as well, and scale them to very large training sets.

Map-reduce and summation over the training set

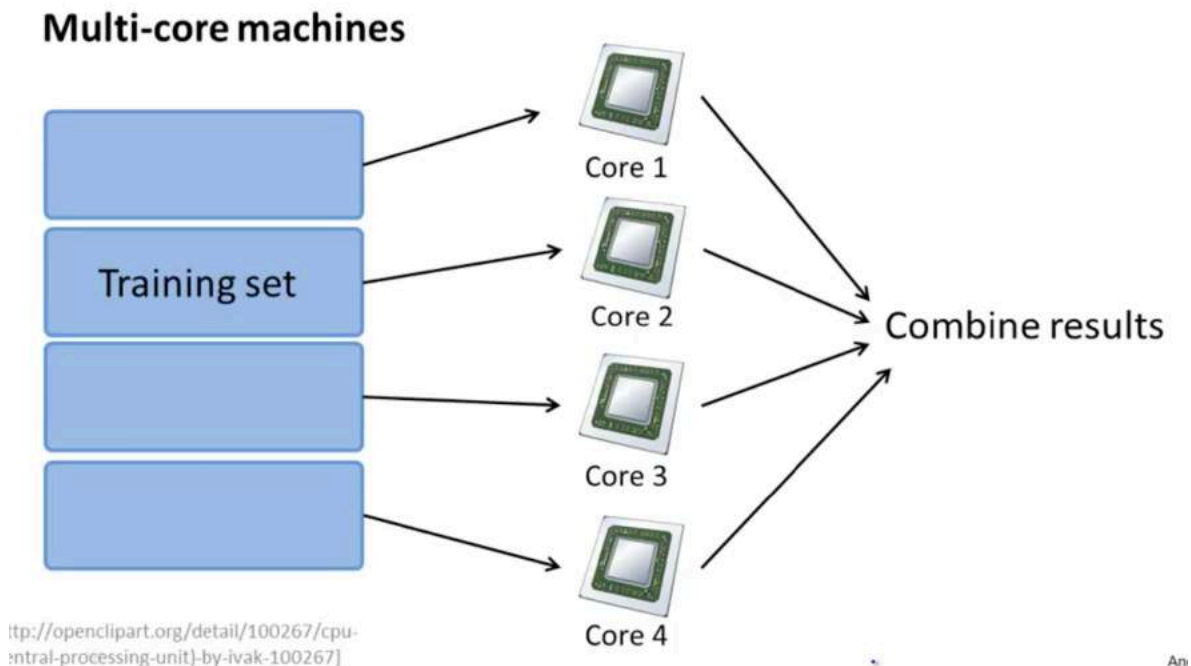
Many learning algorithms can be expressed as computing sums of functions over the training set.

E.g. for advanced optimization, with logistic regression, need:

$$\begin{aligned} \rightarrow \underline{J_{train}(\theta)} &= -\frac{1}{m} \sum_{i=1}^m \underline{y^{(i)} \log h_{\theta}(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))} \\ \rightarrow \underline{\frac{\partial}{\partial \theta_j} J_{train}(\theta)} &= \frac{1}{m} \sum_{i=1}^m \underline{(h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}} \end{aligned}$$

Finally, as one last comment, so far we have been discussing MapReduce algorithms as allowing you to parallelize over multiple computers, maybe multiple computers in a computer cluster or over multiple computers in the data center. It turns out that sometimes even if you have just a single computer, MapReduce can also be applicable. In particular, on many single computers now, you can have multiple processing cores. You can have multiple CPUs, and within each CPU you can have multiple proc cores. If you have a large training set, what you can do if, say, you have a computer with 4 computing cores, what you can do is, even on a single computer you can split the training sets into pieces and send the training set to different cores within a single box, like within a single desktop computer or a single server and use MapReduce this way to divvy up work load. Each of the cores can then carry out the sum over, say, one quarter of your training set, and then they can take the partial sums and combine them, in order to get the summation over the entire training set. The advantage of thinking about MapReduce this way, as paralyzing over cause within a single machine, rather than parallelizing over multiple machines is that, this way you don't have to worry about network latency, because all the communication, all the sending of the [xx] back and forth, all that happens within a single machine. And so network latency becomes much less of an issue compared to if you were using this to over different computers within the data sensor. Finally, one last caveat on parallelizing within a multi-core machine. Depending on the details of your implementation, if you have a multi-core machine and if you have certain numerical linear algebra libraries. It turns out that the sum numerical linear algebra libraries that can automatically parallelize their linear algebra operations across multiple cores within the machine. So if you're fortunate enough to be using one of those numerical linear algebra libraries and

certainly this does not apply to every single library. If you're using one of those libraries and. If you have a very good vectorizing implementation of the learning algorithm. Sometimes you can just implement you standard learning algorithm in a vectorized fashion and not worry about parallelization and numerical linear algebra libraries could take care of some of it for you. So you don't need to implement [xx] but. for other any problems, taking advantage of this sort of map reducing commentation, finding and using this MapReduce formulation and to paralelize a cross coarse except yourself might be a good idea as well and could let you speed up your learning algorithm.



Question

Suppose you apply the map-reduce method to train a neural network on ten machines. In each iteration, what will each of the machines do?

- ☐ Compute either forward propagation or back propagation on 1/5 of the data.
- ☒ Compute forward propagation and back propagation on 1/10 of the data to compute the derivative with respect to that 1/10 of the data.

Correct

- ☐ Compute only forward propagation on 1/10 of the data. (The centralized machine then performs back propagation on all the data).
- ☐ Compute back propagation on 1/10 of the data (after the centralized machine has computed forward propagation on all of the data).

In this video, we talked about the MapReduce approach to parallelizing machine learning by taking a data and spreading them across many computers in the data center. Although these ideas are critical to paralysing across multiple cores within a single computer as well. Today there are some good open source implementations of MapReduce, so there are many users in open source system called Hadoop and using either your own implementation or using someone else's open source implementation, you can use these ideas to parallelize learning algorithms and get them to run on much larger data sets than is possible using just a single machine.

Review

Quiz

1. Suppose you are training a logistic regression classifier using stochastic gradient descent. You find that the cost (say, $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$), averaged over the last 500 examples, plotted as a function of the number of iterations, is slowly increasing over time. Which of the following changes are likely to help?
 - ☐ Try averaging the cost over a larger number of examples (say 1000 examples instead of 500) in the plot.
 - ☐ Try using a larger learning rate α .
 - ☒ Try using a smaller learning rate α .
 - ☐ This is not an issue, as we expect this to occur with stochastic gradient descent.

2. Which of the following statements about stochastic gradient

descent are true? Check all that apply.

- ☒ Before running stochastic gradient descent, you should randomly shuffle (reorder) the training set.
- ☐ One of the advantages of stochastic gradient descent is that it uses parallelization and thus runs much faster than batch gradient descent.
- ☐ In order to make sure stochastic gradient descent is converging, we typically compute $J_{\text{train}}(\theta)$ after each iteration (and plot it) in order to make sure that the cost function is generally decreasing.
- ☒ If you have a huge training set, then stochastic gradient descent may be much faster than batch gradient descent.

3. Which of the following statements about online learning are true? Check all that apply.

- ☒ In the approach to online learning discussed in the lecture video, we repeatedly get a single training example, take one step of stochastic gradient descent using that example, and then move on to the next example.
- ☐ One of the disadvantages of online learning is that it requires a large amount of computer memory/disk space to store all the training examples we have seen.
- ☐ One of the advantages of online learning is that there is no need to pick a learning rate α .
- ☒ When using online learning, in each step we get a new example (x, y) , perform one step of (essentially stochastic gradient descent) learning on that example, and then discard that example and move on to the next.

4. Assuming that you have a very large training set, which of the following algorithms do you think can be parallelized using map-reduce and splitting the training set across different machines? Check all that apply.
- ☒ A neural network trained using batch gradient descent.
 - ☒ Linear regression trained using batch gradient descent.
 - ☐ An online learning setting, where you repeatedly get a single example (x, y) , and want to learn from that single example before moving on.
 - ☐ Logistic regression trained using stochastic gradient descent.
5. Which of the following statements about map-reduce are true? Check all that apply.
- ☐ Running map-reduce over N computers requires that we split the training set into N^2 pieces.
 - ☒ When using map-reduce with gradient descent, we usually use a single machine that accumulates the gradients from each of the map-reduce machines, in order to compute the parameter update for that iteration.
 - ☒ If you have just 1 computer, but your computer has multiple CPUs or multiple cores, then map-reduce might be a viable way to parallelize your learning algorithm.
 - ☒ In order to parallelize a learning algorithm using map-reduce, the first step is to figure out how to express the main work done by the algorithm as computing sums of functions of training examples.

WEEK 11

Application Example: Photo OCR

Photo OCR

Problem Description and Pipeline

In this and the next few videos, I want to tell you about a machine learning application example, or a machine learning application history centered around an application called Photo OCR . There are three reasons why I want to do this, first I wanted to show you an example of how a complex machine learning system can be put together. Second, once told the concepts of a machine learning a type line and how to allocate resources when you're trying to decide what to do next. And this can either be in the context of you working by yourself on the big application Or it can be the context of a team of developers trying to build a complex application together. And then finally, the Photo OCR problem also gives me an excuse to tell you about just a couple more interesting ideas for machine learning. One is some ideas of how to apply machine learning to computer vision problems, and second is the idea of artificial data synthesis, which we'll see in a couple of videos. So, let's start by talking about what is the Photo OCR problem. Photo OCR stands for Photo Optical Character Recognition. With the growth of digital photography and more recently the growth of camera in our cell phones we now have tons of visual pictures that we take all over the place. And one of the things that has interested many developers is how to get our computers to understand the content of these pictures a little bit better. The photo OCR problem focuses on how to get computers to read the text to the purest in images that we take. Given an image like this it might be nice if a computer can read the text in this

image so that if you're trying to look for this picture again you type in the words, lulu bees and and have it automatically pull up this picture, so that you're not spending lots of time digging through your photo collection Maybe hundreds of thousands of pictures in. The Photo OCR problem does exactly this, and it does so in several steps. First, given the picture it has to look through the image and detect where there is text in the picture. And after it has done that or if it successfully does that it then has to look at these text regions and actually read the text in those regions, and hopefully if it reads it correctly, it'll come up with these transcriptions of what is the text that appears in the image. Whereas OCR, or optical character recognition of scanned documents is relatively easier problem, doing OCR from photographs today is still a very difficult machine learning problem, and you can do this. Not only can this help our computers to understand the content of our though images better, there are also applications like helping blind people, for example, if you could provide to a blind person a camera that can look at what's in front of them, and just tell them the words that my be on the street sign in front of them. With car navigation systems. For example, imagine if your car could read the street signs and help you navigate to your destination.

The Photo OCR problem



In order to perform photo OCR, here's what we can do. First we can go through the image and find the regions where there's text and image. So, shown here is one example of text and image that the photo OCR system may find. Second, given the rectangle around that text region, we can then do character segmentation, where we might take this text box that says "Antique Mall" and try to segment it out into the locations of the individual characters. And finally, having segmented out into individual characters, we can then run a crossfire, which looks at the images of the visual characters, and tries to figure out the

first character's an A, the second character's an N, the third character is a T, and so on, so that up by doing all this how that hopefully you can then figure out that this phrase is Rulegee's antique mall and similarly for some of the other words that appear in that image. I should say that there are some photo OCR systems that do even more complex things, like a bit of spelling correction at the end. So if, for example, your character segmentation and character classification system tells you that it sees the word c 1 e a n i n g. Then, you know, a sort of spelling correction system might tell you that this is probably the word 'cleaning', and your character classification algorithm had just mistaken the l for a 1. But for the purpose of what we want to do in this video, let's ignore this last step and just focus on the system that does these three steps of text detection, character segmentation, and character classification. A system like this is what we call a machine learning pipeline.

Photo OCR pipeline

→ 1. Text detection



→ 2. Character segmentation



→ 3. Character classification



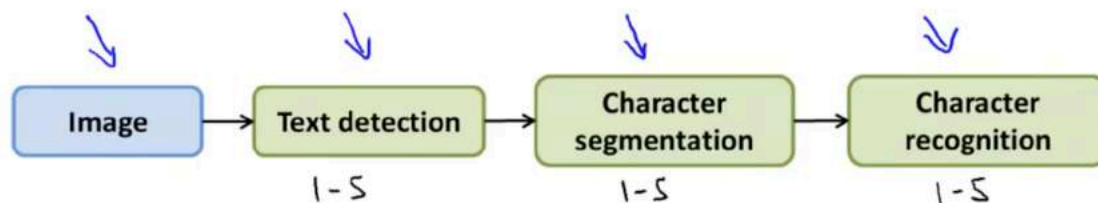
Cleaning → ~~Cleaning~~

Andrew

In particular, here's a picture showing the photo OCR pipeline. We have an image, which then fed to the text detection system text regions, we then segment out the characters--the individual characters in the text--and then finally we recognize the individual characters. In many complex machine learning systems, these sorts of pipelines are common, where you can have multiple modules--in this example, the text detection, character segmentation, character recognition modules--each of which may be machine learning component, or sometimes it may not be a machine learning component but to have a set of modules that act one after another on some piece of data in order to produce the output you want, which in the photo OCR example is to find the transcription of the text that appeared in the image. If you're designing a machine learning system one of the most important decisions will often be what exactly is the pipeline that you want to put together. In other words, given the photo OCR problem, how do you break this problem down into a sequence of different modules. And you design the pipeline and each the performance of

each of the modules in your pipeline. will often have a big impact on the final performance of your algorithm. If you have a team of engineers working on a problem like this is also very common to have different individuals work on different modules. So I could easily imagine tech easily being the of anywhere from 1 to 5 engineers, character segmentation maybe another 1-5 engineers, and character recognition being another 1-5 engineers, and so having a pipeline like often offers a natural way to divide up the workload amongst different members of an engineering team, as well. Although, or course, all of this work could also be done by just one person if that's how you want to do it. In complex machine learning systems the idea of a pipeline, of a machine of a pipeline, is pretty pervasive.

Photo OCR pipeline



Question

When someone refers to a “machine learning pipeline,” he or she is referring to:

- ☐ A PhotoOCR system.
- ☐ A character recognition system.
- ☒ A system with many stages / components, several of which may use machine learning.
- ☐ An application in plumbing. (Haha.)

And what you just saw is a specific example of how a Photo OCR pipeline might work. In the next few videos I'll tell you a little bit more about this pipeline, and we'll continue to use this as an example to illustrate--I think--a few more key concepts of machine learning.

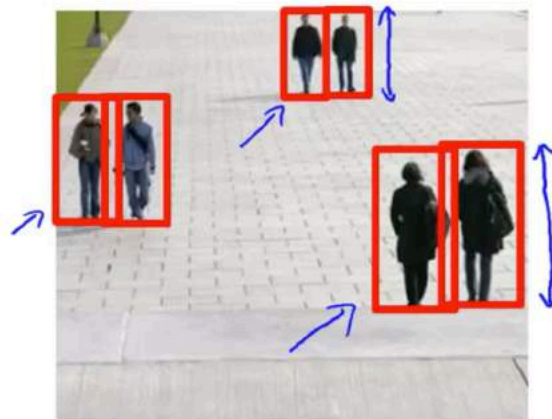
Sliding Windows

In the previous video, we talked about the photo OCR pipeline and how that worked. In which we would take an image and pass the Through a sequence of machine learning components in order to try to read the text that appears in an image. In this video I like to. A little bit more about how the individual components of the pipeline works. In particular most of this video will center around the discussion. of whats called a sliding windows. The first stage of the filter was the Text detection where we look at an image like this and try to find the regions of text that appear in this image. Text detection is an unusual problem in computer vision. Because depending on the length of the text you're trying to find, these rectangles that you're trying to find can have different aspect. So in order to talk about detecting things in images let's start with a simpler example of pedestrian detection and we'll then later go back to. Ideas that were developed in pedestrian detection and apply them to text detection. So in pedestrian detection you want to take an image that looks like this and the whole idea is the individual pedestrians that appear in the image. So there's one pedestrian that we found, there's a second one, a third one a fourth one, a fifth one. And a one. This problem is maybe slightly simpler than text detection just for the reason that the aspect ratio of most pedestrians are pretty similar. Just using a fixed aspect ratio for these rectangles that we're trying to find. So by aspect ratio I mean the ratio between the height and the width of these rectangles. They're all the same. for different pedestrians but for text detection the height and width ratio is different for different lines of text Although for pedestrian detection, the pedestrians can be different distances away from the camera and so the height of these rectangles can be different depending on how far away they are. but the aspect ratio is the same.

Text detection



Pedestrian detection



In order to build a pedestrian detection system here's how you can go about it. Let's say that we decide to standardize on this aspect ratio of 82 by 36 and we could have chosen some rounded number like 80 by 40 or something, but 82 by 36 seems alright. What we would do is then go out and collect large training sets of positive and negative examples. Here are examples of 82 X 36 image patches that do contain pedestrians and here are examples of images that do not. On this slide I show 12 positive examples of $y=1$ and 12 examples of $y=0$. In a more typical pedestrian detection application, we may have anywhere from a 1,000 training examples up to maybe 10,000 training examples, or even more if you can get even larger training sets. And what you can do, is then train in your network or some other learning algorithm to take this input, an MS patch of dimension 82 by 36, and to classify 'y' and to classify that image patch as either containing a pedestrian or not. So this gives you a way of applying supervised learning in order to take an image patch can determine whether or not a pedestrian appears in that image capture.

Supervised learning for pedestrian detection

x = pixels in 82x36 image patches



Positive examples ($y = 1$)



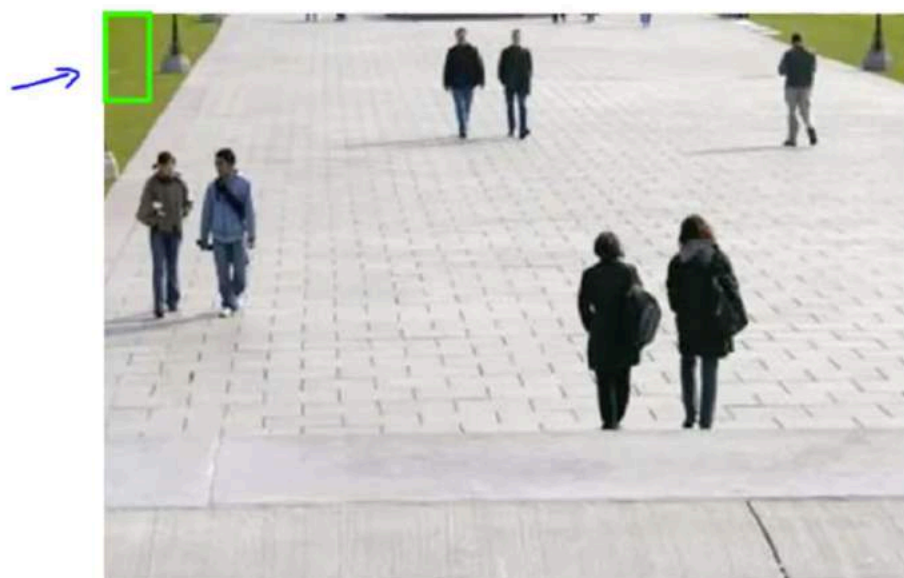
Negative examples ($y = 0$)

1,000
10,000
...

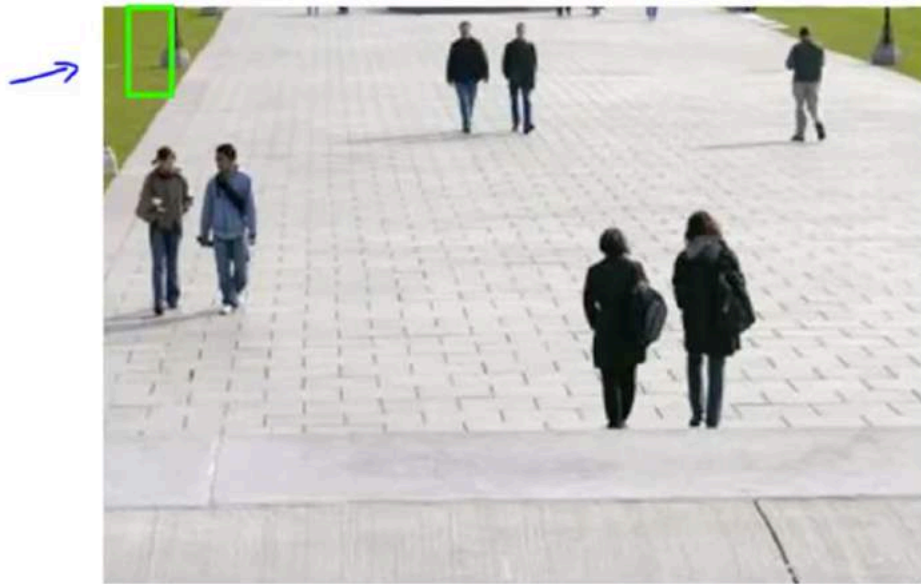
Now, let's say we get a new image, a test set image like this and we want to try to find a pedestrian's picture image. What we would do is start by taking a rectangular patch of this image. Like that shown up here, so that's maybe a 82 X 36 patch of this image, and run that image patch through our classifier to determine whether or not there is a pedestrian in that image patch, and hopefully our classifier will return y equals 0 for that patch, since there is no pedestrian. Next, we then take that green rectangle and we slide it over a bit and then run that new image patch through our classifier to decide if there's a pedestrian there. And having done that, we then slide the window further to

the right and run that patch through the classifier again. The amount by which you shift the rectangle over each time is a parameter, that's sometimes called the step size of the parameter, sometimes also called the slide parameter, and if you step this one pixel at a time. So you can use the step size or stride of 1, that usually performs best, that is more cost effective, and so using a step size of maybe 4 pixels at a time, or eight pixels at a time or some large number of pixels might be more common, since you're then moving the rectangle a little bit more each time. So, using this process, you continue stepping the rectangle over to the right a bit at a time and running each of these patches through a classifier, until eventually, as you slide this window over the different locations in the image, first starting with the first row and then we go further rows in the image, you would then run all of these different image patches at some step size or some stride through your classifier. Now, that was a pretty small rectangle, that would only detect pedestrians of one specific size. What we do next is start to look at larger image patches. So now let's take larger images patches, like those shown here and run those through the crossfire as well.

Sliding window detection

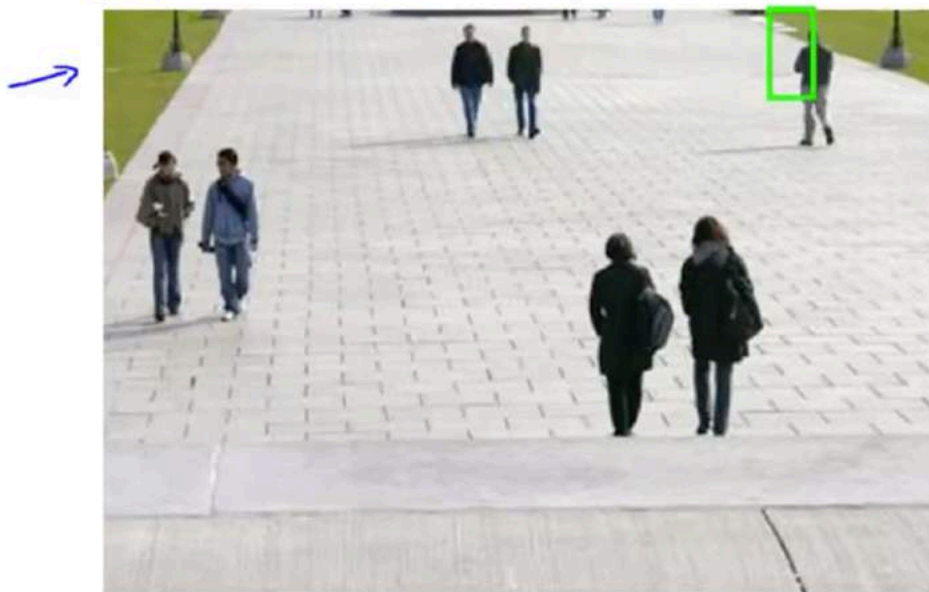


Sliding window detection



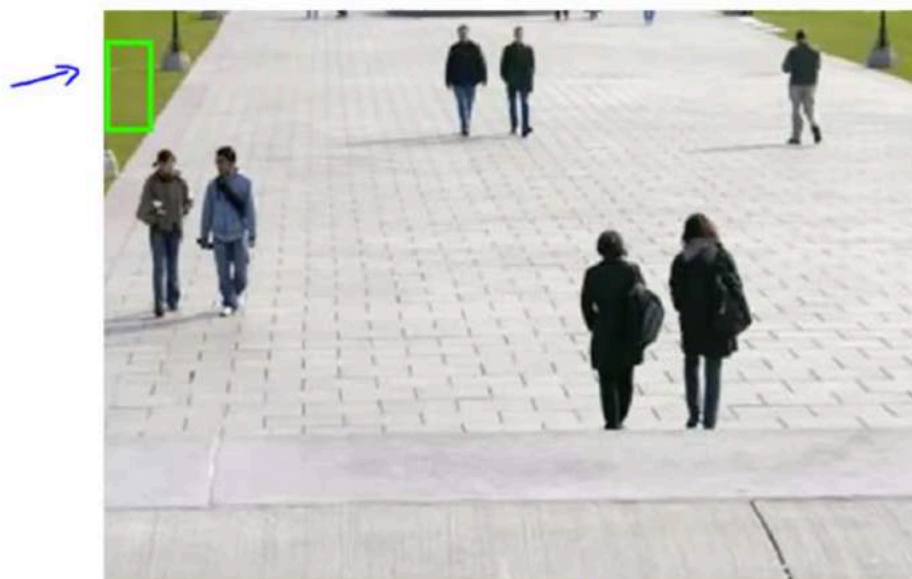
Sliding window detection

step-size / stride
↔



Sliding window detection

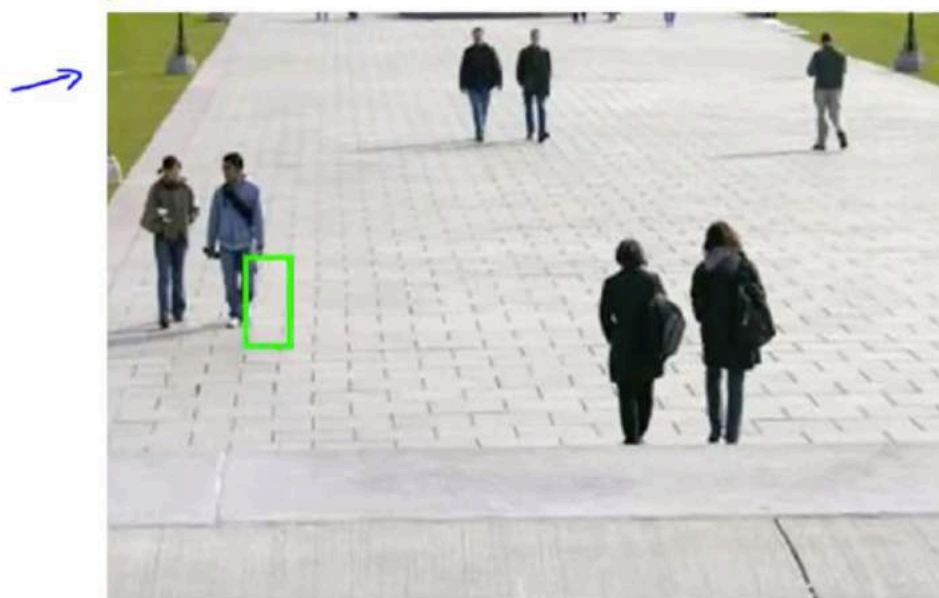
step-size / stride
↔



src: OpenCV

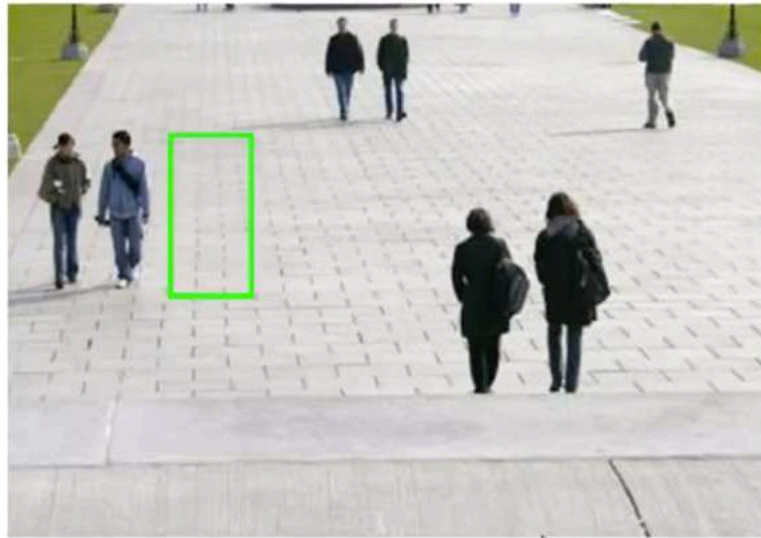
Sliding window detection

step-size / stride
↔



src: OpenCV

Sliding window detection



Sliding window detection



And by the way when I say take a larger image patch, what I really mean is when you take an image patch like this, what you're really doing is taking that image patch, and resizing it down to 82 X 36, say. So you take this larger patch and re-size it to be smaller image and then it would be the smaller size image that is what you would pass through your classifier to try and decide if

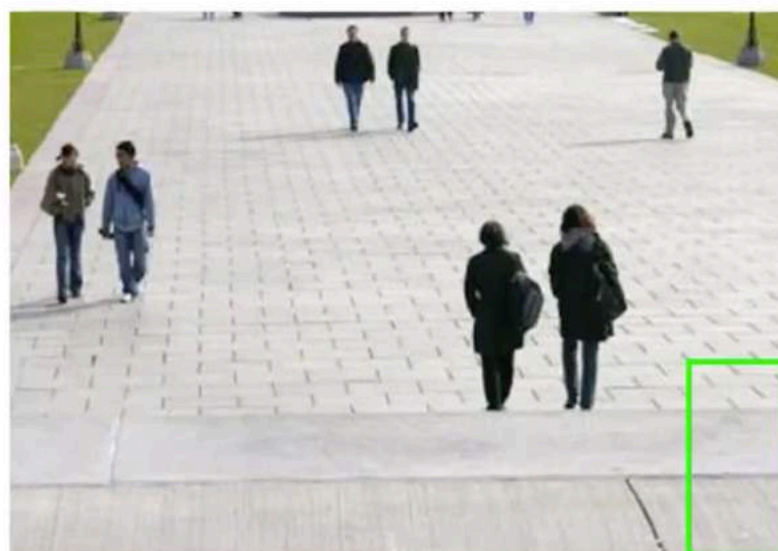
there is a pedestrian in that patch. And finally you can do this at an even larger scales and run that side of Windows to the end

Sliding window detection



© 2014 Intel

Sliding window detection

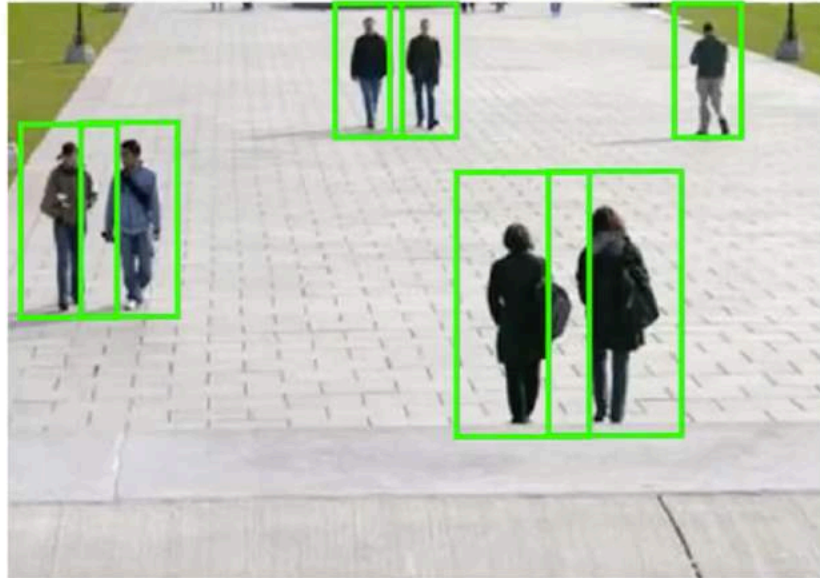


© 2014 Intel

And after this whole process hopefully your algorithm will detect whether there's a pedestrian appears in the image, so that's how you train a classifier,

and then use a sliding windows classifier, or use a sliding windows detector in order to find pedestrians in the image.

Sliding window detection



Let's have a turn to the text detection example and talk about that stage in our photo OCR pipeline, where our goal is to find the text regions in unit.

Text detection



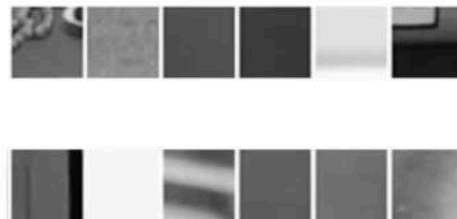
Similar to pedestrian detection you can come up with a label training set with

positive examples and negative examples with examples corresponding to regions where text appears. So instead of trying to detect pedestrians, we're now trying to detect texts. And so positive examples are going to be patches of images where there is text. And negative examples is going to be patches of images where there isn't text. Having trained this we can now apply it to a new image, into a test set image.

Text detection



Positive examples ($y = 1$)



Negative examples ($y = 0$)

Now, last time we run, for this example we are going to run a sliding windows at just one fixed scale just for purpose of illustration, meaning that I'm going to use just one rectangle size. But lets say I run my little sliding windows classifier on lots of little image patches like this if I do that, what I'll end up with is a result like this where the white region show where my text detection system has found text and so the axis' of these two figures are the same. So there is a region up here, of course also a region up here, so the fact that this black up here represents that the classifier does not think it's found any texts up there, whereas the fact that there's a lot of white stuff here, that reflects that classifier thinks that it's found a bunch of texts. over there on the image. What i have done on this image on the lower left is actually use white to show where the classifier thinks it has found text. And different shades of grey correspond to the probability that was output by the classifier, so like the shades of grey corresponds to where it thinks it might have found text but has lower confidence the bright white response to whether the classifier, up with a very high probability, estimated probability of there being pedestrians in that location. We aren't quite done yet because what we actually want to do is draw rectangles around all the region where this text in the image, so were going to take one more step which is we take the output of the classifier and apply to it what is called an expansion operator. So what that does is, it take the image here, and it takes each of the white blobs, it takes each of the white regions and it expands that white region. Mathematically, the way you implement that

is, if you look at the image on the right, what we're doing to create the image on the right is, for every pixel we are going to ask, is it within some distance of a white pixel in the left image. And so, if a specific pixel is within, say, five pixels or ten pixels of a white pixel in the leftmost image, then we'll also color that pixel white in the rightmost image. And so, the effect of this is, we'll take each of the white blobs in the leftmost image and expand them a bit, grow them a little bit, by seeing whether the nearby pixels, the white pixels, and then coloring those nearby pixels in white as well. Finally, we are just about done. We can now look at this rightmost image and just look at the connecting components and look at the as white regions and draw bounding boxes around them. And in particular, if we look at all the white regions, like this one, this one, this one, and so on, and if we use a simple heuristic to rule out rectangles whose aspect ratios look funny because we know that boxes around text should be much wider than they are tall. And so if we ignore the thin, tall blobs like this one and this one, and we discard these ones because they are too tall and thin, and we then draw a the rectangles around the ones whose aspect ratio that's a height to what ratio looks like for text regions, then we can draw rectangles, the bounding boxes around this text region, this text region, and that text region, corresponding to the Lula B's antique mall logo, the Lula B's, and this little open sign. Of over there. This example by the actually misses one piece of text. This is very hard to read, but there is actually one piece of text there. That says [xx] are corresponding to this but the aspect ratio looks wrong so we discarded that one. So you know it's ok on this image, but in this particular example the classifier actually missed one piece of text. It's very hard to read because there's a piece of text written against a transparent window. So that's text detection using sliding windows. And having found these rectangles with the text in it, we can now just cut out these image regions and then use later stages of pipeline to try to meet the texts.



Question

Suppose you are running a text detector using 20x20 image patches. You run the classifier on a 200x200 image and when using sliding window, you "step" the detector by 4 pixels each time. (For this problem assume you apply the algorithm at only one scale.) About how many times will you end up running your classifier on a single image? (Pick the closest answer.)

- ☐ About 100 times.
- ☐ About 400 times.
- ☒ About 2,500 times.

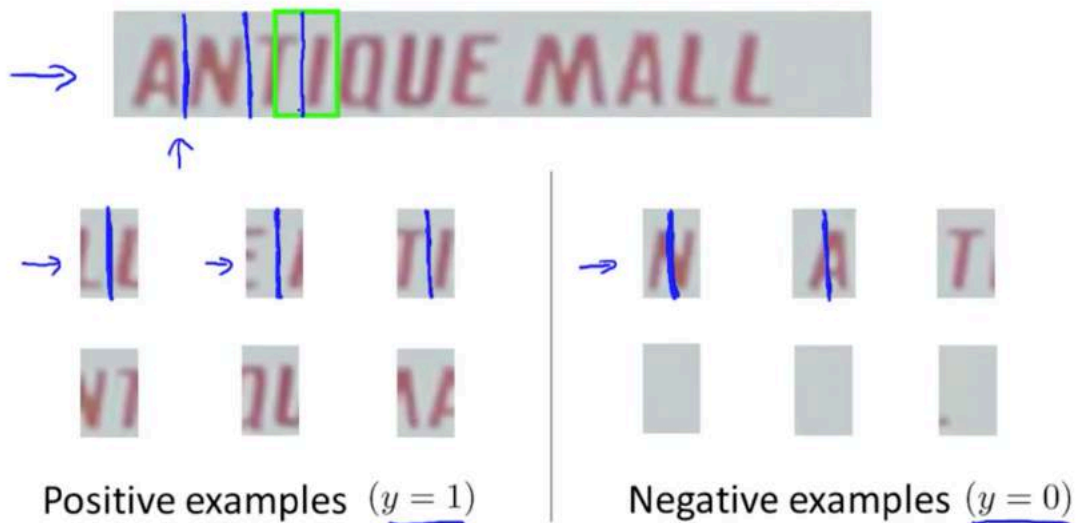
Correct

- ☐ About 40,000 times.

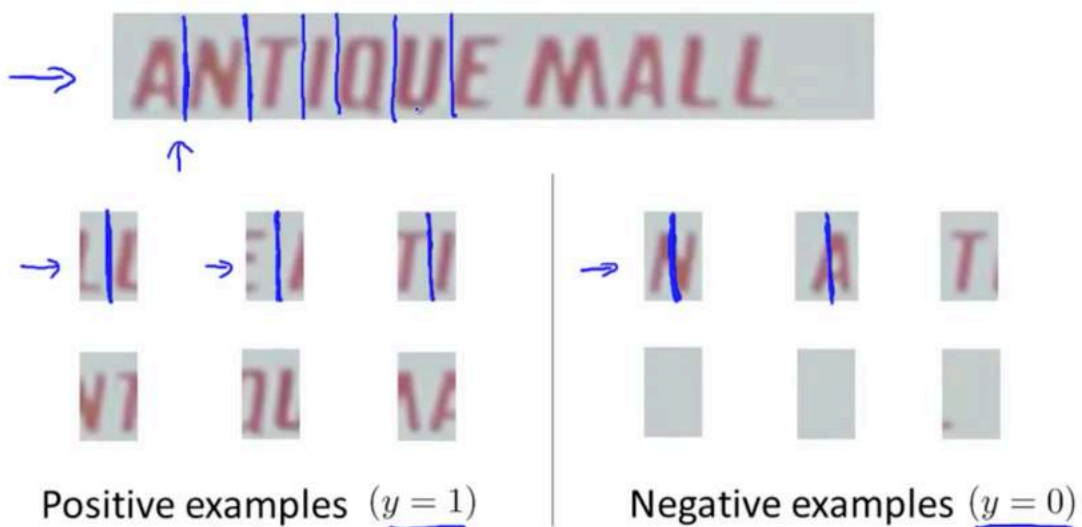
Now, you recall that the second stage of pipeline was character segmentation, so given an image like that shown on top, how do we segment out the individual characters in this image? So what we can do is again use a supervised learning algorithm with some set of positive and some set of negative examples, what we're going to do is look in the image patch and try to decide if there is a split between two characters right in the middle of that image patch. So for initial positive examples. This first cross example, this image patch looks like the middle of it is indeed the middle has splits between two characters and the second example again this looks like a positive example, because if I split two characters by putting a line right down the middle, that's the right thing to do. So, these are positive examples, where the middle of the image represents a gap or a split between two distinct characters, whereas the negative examples, well, you know, you don't want to split two characters right in the middle, and so these are negative examples because they don't represent the midpoint between two characters. So what we will do is, we will train a classifier, maybe using a new network, maybe using a different learning algorithm, to try to classify between the positive and negative examples. Having trained such a classifier, we can then run this on this sort of text that our text detection system has pulled out. As we start by looking at that rectangle, and we ask, "Gee, does it look like the middle of that green rectangle, does it look like the midpoint between two characters?". And hopefully, the classifier will say no, then we slide the window over and this is a one dimensional sliding window classifier, because we're going to slide the window only in one straight line from left to right, there's no different rows here. There's only one row here. But now, with the classifier in this position, we ask, well, should we split those two characters or should we put a split right down the middle of this rectangle. And hopefully, the classifier will output y equals

one, in which case we will decide to draw a line down there, to try to split two characters. Then we slide the window over again, optic process, don't close the gap, slide over again, optic says yes, do split there and so on, and we slowly slide the classifier over to the right and hopefully it will classify this as another positive example and so on. And we will slide this window over to the right, running the classifier at every step, and hopefully it will tell us, you know, what are the right locations to split these characters up into, just split this image up into individual characters. And so that's 1D sliding windows for character segmentation.

1D Sliding window for character segmentation



1D Sliding window for character segmentation



So, here's the overall photo OCR pipe line again. In this video we've talked

about the text detection step, where we use sliding windows to detect text. And we also use a one-dimensional sliding windows to do character segmentation to segment out, you know, this text image in division of characters. The final step through the pipeline is the character qualification step and that step you might already be much more familiar with the early videos on supervised learning where you can apply a standard supervised learning within maybe on your network or maybe something else in order to take it's input, an image like that and classify which alphabet or which 26 characters A to Z, or maybe we should have 36 characters if you have the numerical digits as well, the multi class classification problem where you take it's input and image contained a character and decide what is the character that appears in that image

Photo OCR pipeline

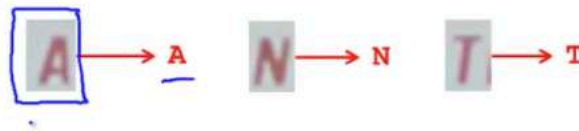
→ 1. Text detection



→ 2. Character segmentation



→ 3. Character classification



So that was the photo OCR pipeline and how you can use ideas like sliding windows classifiers in order to put these different components to develop a photo OCR system. In the next few videos we keep on using the problem of photo OCR to explore somewhat interesting issues surrounding building an application like this.

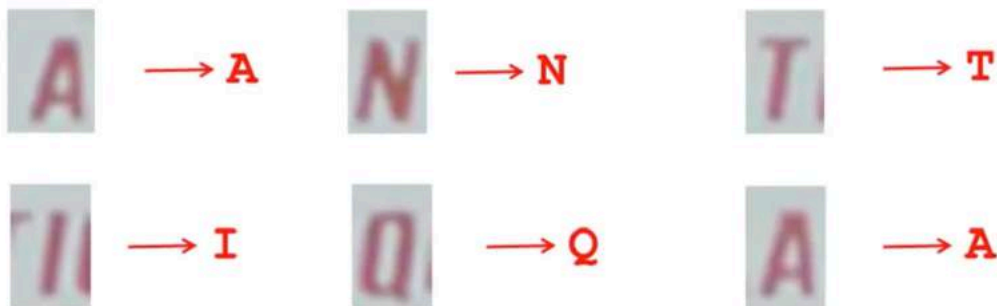
Getting Lots of Data and Artificial Data

I've seen over and over that one of the most reliable ways to get a high performance machine learning system is to take a low bias learning algorithm and to train it on a massive training set. But where did you get so much training

data from? Turns out that the machine learning there's a fascinating idea called artificial data synthesis, this doesn't apply to every single problem, and to apply to a specific problem, often takes some thought and innovation and insight. But if this idea applies to your machine learning problem, it can sometimes be an easy way to get a huge training set to give to your learning algorithm. The idea of artificial data synthesis comprises of two variations, mainly the first is if we are essentially creating data from scratch, creating new data from scratch. And the second is if we already have a small labeled training set and we somehow have to amplify that training set or use a small training set to turn that into a larger training set and in this video we'll go over both those ideas.

To talk about the artificial data synthesis idea, let's use the character portion of the photo OCR pipeline, we want to take its input image and recognize what character it is.

Character recognition



If we go out and collect a large labeled data set, here's what it is and what it looks like. For this particular example, I've chosen a square aspect ratio. So we're taking square image patches. And the goal is to take an image patch and recognize the character in the middle of that image patch. And for the sake of simplicity, I'm going to treat these images as grey scale images, rather than color images. It turns out that using color doesn't seem to help that much for this particular problem. So given this image patch, we'd like to recognize that that's a T. Given this image patch, we'd like to recognize that it's an 'S'. Given that image patch we would like to recognize that as an 'I' and so on. So all of these, our examples of row images, how can we come up with a much larger training set? Modern computers often have a huge font library and if you use a word processing software, depending on what word processor you use, you might have all of these fonts and many, many more already stored inside. And, in fact, if you go to different websites, there are, again, huge, free font libraries on the internet we can download many, many different types of fonts, hundreds or perhaps thousands of different fonts. So if you want more training examples,

one thing you can do is just take characters from different fonts and paste these characters against different random backgrounds. So you might take this ---- and paste that c against a random background. If you do that you now have a training example of an image of the character C.

Artificial data synthesis for photo OCR



Real data



So after some amount of work, you know this, and it is a little bit of work to synthesize realistic looking data. But after some amount of work, you can get a synthetic training set like that. Every image shown on the right was actually a synthesized image. Where you take a font, maybe a random font downloaded off the web and you paste an image of one character or a few characters from that font against this other random background image. And then apply maybe a little blurring operators -----of app finder, distortions that app finder, meaning just the sharing and scaling and little rotation operations and if you do that you get a synthetic training set, on what the one shown here. And this is work, grade, it is, it takes thought at work, in order to make the synthetic data look realistic, and if you do a sloppy job in terms of how you create the synthetic data then it actually won't work well. But if you look at the synthetic data looks remarkably similar to the real data. And so by using synthetic data you have essentially an unlimited supply of training examples for artificial training synthesis And so, if you use this source synthetic data, you have essentially unlimited supply of label data to create a improvised learning algorithm for the character recognition problem. So this is an example of artificial data synthesis where youre basically creating new data from scratch, you just generating brand new images from scratch.

Artificial data synthesis for photo OCR



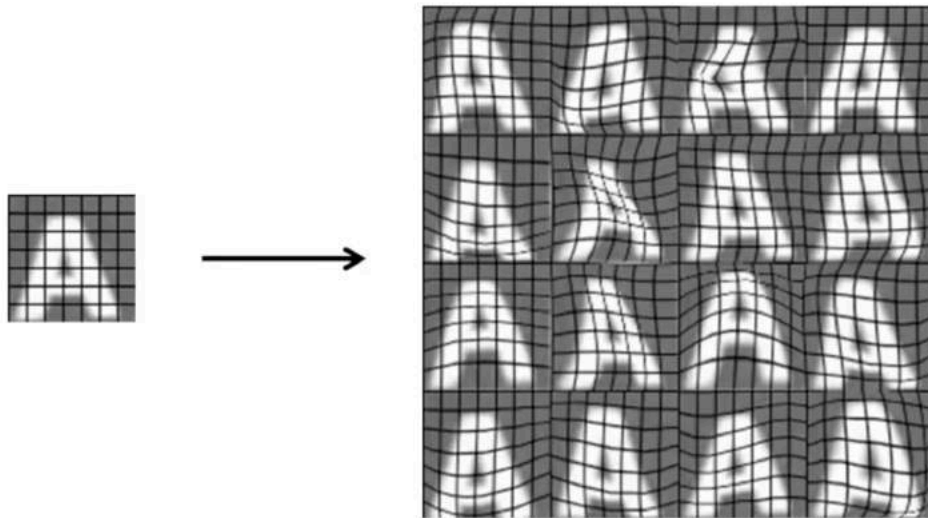
Real data



Synthetic data



The other main approach to artificial data synthesis is where you take a examples that you currently have, that we take a real example, maybe from real image, and you create additional data, so as to amplify your training set. So here is an image of a compared to a from a real image, not a synthesized image, and I have overlayed this with the grid lines just for the purpose of illustration. Actually have these ----. So what you can do is then take this alphabet here, take this image and introduce artificial warpings[sp?] or artificial distortions into the image so they can take the image a and turn that into 16 new examples. So in this way you can take a small label training set and amplify your training set to suddenly get a lot more examples, all of it. Again, in order to do this for application, it does take thought and it does take insight to figure out what our reasonable sets of distortions, or whether these are ways that amplify and multiply your training set, and for the specific example of character recognition, introducing these warping seems like a natural choice, but for a different learning machine application, there may be different the distortions that might make more sense.


Synthesizing data by introducing distortions





Let me just show one example from the totally different domain of speech recognition. So the speech recognition, let's say you have audio clips and you want to learn from the audio clip to recognize what were the words spoken in that clip. So let's see how one labeled training example. So let's say you have one labeled training example, of someone saying a few specific words. So let's play that audio clip here. 0-1-2-3-4-5. Alright, so someone counting from 0 to 5, and so you want to try to apply a learning algorithm to try to recognize the words said in that. So, how can we amplify the data set? Well, one thing we do is introduce additional audio distortions into the data set. So here I'm going to add background sounds to simulate a bad cell phone connection. When you hear beeping sounds, that's actually part of the audio track, that's nothing wrong with the speakers, I'm going to play this now. 0-1-2-3-4-5. Right, so you can listen to that sort of audio clip and recognize the sounds, that seems like another useful training example to have, here's another example, noisy background. Zero, one, two, three four five you know of cars driving past, people walking in the background, here's another one, so taking the original clean audio clip so taking the clean audio of someone saying 0 1 2 3 4 5 we can then automatically synthesize these additional training examples and thus amplify one training example into maybe four different training examples. So let me play this final example, as well. 0-1 3-4-5 So by taking just one labelled example, we have to go through the effort to collect just one labelled example fall of the 01205, and by synthesizing additional distortions, by introducing different background sounds, we've now multiplied this one example into many more examples. Much work by just automatically adding these different background sounds to the clean audio

Synthesizing data by introducing distortions: Speech recognition

 Original audio: 

 Audio on bad cellphone connection

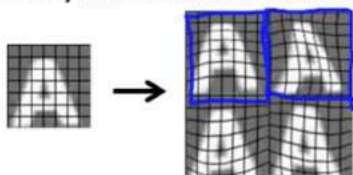
 Noisy background: Crowd

 Noisy background: Machinery

Just one word of warning about synthesizing data by introducing distortions: if you try to do this yourself, the distortions you introduce should be representative of the source of noises, or distortions, that you might see in the test set. So, for the character recognition example, you know, the working things being introduced are actually kind of reasonable, because an image A that looks like that, that's, could be an image that we could actually see in a test set. Reflect a fact And, you know, that image on the upper-right, that could be an image that we could imagine seeing. And for audio, well, we do want to recognize speech, even against a bad self internal connection, against different types of background noise, and so for the audio, we're again synthesizing examples are actually representative of the sorts of examples that we want to classify, that we want to recognize correctly. In contrast, usually it does not help perhaps you actually a meaning as noise to your data. I'm not sure you can see this, but what we've done here is taken the image, and for each pixel, in each of these 4 images, has just added some random Gaussian noise to each pixel. To each pixel, is the pixel brightness, it would just add some, you know, maybe Gaussian random noise to each pixel. So it's just a totally meaningless noise, right? And so, unless you're expecting to see these sorts of pixel wise noise in your test set, this sort of purely random meaningless noise is less likely to be useful. But the process of artificial data synthesis it is you know a little bit of an art as well and sometimes you just have to try it and see if it works. But if you're trying to decide what sorts of distortions to add, you know, do think about what other meaningful distortions you might add that will cause you to generate additional training examples that are at least somewhat representative of the sorts of images you expect to see in your test sets.

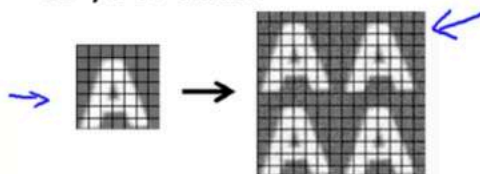
Synthesizing data by introducing distortions

- Distortion introduced should be representation of the type of noise/distortions in the test set.



- Audio:
Background noise,
bad cellphone connection

- Usually does not help to add purely random/meaningless noise to your data.



- x_i = intensity (brightness) of pixel i
→ $x_i \leftarrow x_i + \text{random noise}$

[Adam Coates and Tao Wang]

Question

Suppose you are training a linear regression model with m examples by minimizing:

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

Suppose you duplicate every example by making two identical copies of it. That is, where you previously had one example $(x^{(i)}, y^{(i)})$, you now have two copies of it, so you now have $2m$ examples. Is this likely to help?

- ☐ Yes, because increasing the training set size will reduce variance.
- ☐ Yes, so long as you are using a large number of features (a "low bias" learning algorithm).
- ☐ No. You may end up with different parameters θ , but they are unlikely to do any better than the ones learned from the original training set.
- ☒ No, and in fact you will end up with the same parameters θ as before you duplicated the data.

Correct

Finally, to wrap up this video, I just wanna say a couple of words, more about this idea of getting loss of data via artificial data synthesis. As always, before expending a lot of effort, you know, figuring out how to create artificial training

examples, it's often a good practice is to make sure that you really have a low biased crossfire, and having a lot more training data will be of help. And standard way to do this is to plot the learning curves, and make sure that you only have a low as well, high variance falsifier. Or if you don't have a low bias falsifier, you know, one other thing that's worth trying is to keep increasing the number of features that your classifier has, increasing the number of hidden units in your network, saying, until you actually have a low bias falsifier, and only then, should you put the effort into creating a large, artificial training set, so what you really want to avoid is to, you know, spend a whole week or spend a few months figuring out how to get a great artificially synthesized data set. Only to realize afterward, that, you know, your learning algorithm, performance doesn't improve that much, even when you're given a huge training set. So that's about my usual advice about of a testing that you really can make use of a large training set before spending a lot of effort going out to get that large training set. Second is, when i'm working on machine learning problems, one question I often ask the team I'm working with, often ask my students, which is, how much work would it be to get 10 times as much data as we currently had. When I face a new machine learning application very often I will sit down with a team and ask exactly this question, I've asked this question over and over and over and I've been very surprised how often this answer has been that. You know, it's really not that hard, maybe a few days of work at most, to get ten times as much data as we currently have for a machine running application and very often if you can get ten times as much data there will be a way to make your algorithm do much better. So, you know, if you ever join the product team working on some machine learning application product this is a very good questions ask yourself ask the team don't be too surprised if after a few minutes of brainstorming if your team comes up with a way to get literally ten times this much data, in which case, I think you would be a hero to that team, because with 10 times as much data, I think you'll really get much better performance, just from learning from so much data. So there are several waysand that comprised both the ideas of generating data from scratch using random fonts and so on. As well as the second idea of taking an existing example and and introducing distortions that amplify to enlarge the training set A couple of other examples of ways to get a lot more data are to collect the data or to label them yourself. So one useful calculation that I often do is, you know, how many minutes, how many hours does it take to get a certain number of examples, so actually sit down and figure out, you know, suppose it takes me ten seconds to label one example then and, suppose that, for our application, currently we have 1000 labeled examples examples so ten times as much of that would be if n were equal to ten thousand. A second way to get a lot of data is to just collect the data and you label it yourself. So what I mean by this is I will often set down and do a calculation to figure out how much time, you know just like how many hours will it take, how many hours or how many days will it take for me or for someone else to just sit down and collect ten times as much data, as we have currently, by collecting the data ourselves and labeling them ourselves. So, for example, that, for our machine learning

application, currently we have 1,000 examples, so $M = 1,000$. That what we do is sit down and ask, how long does it take me really to collect and label one example. And sometimes maybe it will take you, you know ten seconds to label one new example, and so if I want 10 X as many examples, I'd do a calculation. If it takes me 10 seconds to get one training example. If I wanted to get 10 times as much data, then I need 10,000 examples. So I do the calculation, how long is it gonna take to label, to manually label 10,000 examples, if it takes me 10 seconds to label 1 example. So when you do this calculation, often I've seen many you would be surprised, you know, how little, or sometimes a few days at work, sometimes a small number of days of work, well I've seen many teams be very surprised that sometimes how little work it could be, to just get a lot more data, and let that be a way to give your learning app to give you a huge boost in performance, and necessarily, you know, sometimes when you've just managed to do this, you will be a hero and whatever product development, whatever team you're working on, because this can be a great way to get much better performance. Third and finally, one sometimes good way to get a lot of data is to use what's now called crowd sourcing. So today, there are a few websites or a few services that allow you to hire people on the web to, you know, fairly inexpensively label large training sets for you. So this idea of crowd sourcing, or crowd sourced data labeling, is something that has, is obviously, like an entire academic literature, has some of it's own complications and so on, pertaining to labeler reliability. Maybe, you know, hundreds of thousands of labelers, around the world, working fairly inexpensively to help label data for you, and that I've just had mentioned, there's this one alternative as well. And probably Amazon Mechanical Turk systems is probably the most popular crowd sourcing option right now. This is often quite a bit of work to get to work, if you want to get very high quality labels, but is sometimes an option worth considering as well. If you want to try to hire many people, fairly inexpensively on the web, our labels launch miles of data for you.

Discussion on getting more data

1. Make sure you have a low bias classifier before expending the effort. (Plot learning curves). E.g. keep increasing the number of features/number of hidden units in neural network until you have a low bias classifier.
2. "How much work would it be to get 10x as much data as we currently have?"

- Artificial data synthesis
- Collect/label it yourself

→ #hours? $m = 1,000$
 → 10 secs/example
 $m = \underline{10,000}$ ←

Discussion on getting more data

1. Make sure you have a low bias classifier before expending the effort. (Plot learning curves). E.g. keep increasing the number of features/number of hidden units in neural network until you have a low bias classifier.
2. "How much work would it be to get 10x as much data as we currently have?"
 - Artificial data synthesis
 - Collect/label it yourself
 - "Crowd source" (E.g. Amazon Mechanical Turk)

Question

You've just joined a product group that has been developing a machine learning application for the last 12 months using 1,000 training examples. Suppose that by manually collecting and labeling examples, it takes you an average of 10 seconds to obtain one extra training example. Suppose you work 8 hours a day. How many days will it take you to get 10,000 examples? (Pick the closest answer.)

- ☐ About 1 day.
- ☒ About 3.5 days.
- ☐ About 28 days.
- ☐ About 200 days.

So this video, we talked about the idea of artificial data synthesis of either creating new data from scratch, looking, using the ramming funds as an example, or by amplifying an existing training set, by taking existing label examples and introducing distortions to it, to sort of create extra label examples. And finally, one thing that I hope you remember from this video this idea of if you are facing a machine learning problem, it is often worth doing two things. One just a sanity check, with learning curves, that having more data would help. And second, assuming that that's the case, I will often seat down and ask yourself seriously: what would it take to get ten times as much creative data as you currently have, and not always, but sometimes, you may be surprised by how easy that turns out to be, maybe a few days, a few weeks

at work, and that can be a great way to give your learning algorithm a huge boost in performance

Ceiling Analysis: What Part of the Pipeline to Work on Next

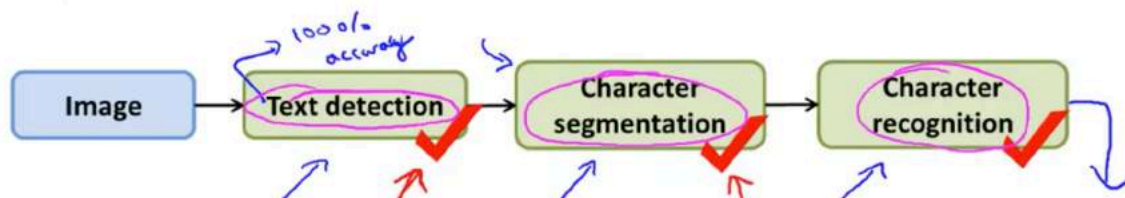
In earlier videos, I've said over and over that, when you're developing a machine learning system, one of the most valuable resources is your time as the developer, in terms of picking what to work on next. Or, if you have a team of developers or a team of engineers working together on a machine learning system. Again, one of the most valuable resources is the time of the engineers or the developers working on the system. And what you really want to avoid is that you or your colleagues your friends spend a lot of time working on some component. Only to realize after weeks or months of time spent, that all that worked just doesn't make a huge difference on the performance of the final system. In this video what I'd like to do is something called ceiling analysis. When you're the team working on the pipeline machine on your system, this can sometimes give you a very strong signal, a very strong guidance on what parts of the pipeline might be the best use of your time to work on.

To talk about ceiling analysis I'm going to keep on using the example of the photo OCR pipeline. And see right here each of these boxes, text detection, character segmentation, character recognition, each of these boxes can have even a small engineering team working on it. Or maybe the entire system is just built by you, either way. But the question is where should you allocate resources? Which of these boxes is most worth your effort of trying to improve the performance of. In order to explain the idea of ceiling analysis, I'm going to keep using the example of our photo OCR pipeline. As I mentioned earlier, each of these boxes here, each of these machines and components could be the work of a small team of engineers, or the whole system could be built by just one person. But the question is, where should you allocate scarce resources? That is, which of these components, which one or two or maybe all three of these components is most worth your time, to try to improve the performance of. So here's the idea of ceiling analysis. As in the development process for other machine learning systems as well, in order to make decisions on what to do for developing the system is going to be very helpful to have a single rolled number evaluation metric for this learning system. So let's say we pick character level accuracy. So if you're given a test set image, what is the fraction of alphabets or characters in a test image that we recognize

correctly? Or you can pick some other single road number evaluation that you could, if you want. But let's say for whatever evaluation measure we pick, we find that the overall system currently has 72% accuracy. So in other words, we have some set of test set images. And from each test set images, we run it through text detection, then character segmentation, then character recognition. And we find that on our test set the overall accuracy of the entire system was 72% on whatever metric you chose. Now here's the idea behind ceiling analysis, which is that we're going to go through, let's say the first module of our machinery pipeline, say text detection. And what we're going to do, is we're going to monkey around with the test set. We're gonna go to the test set. For every test example, which is going to provide it the correct text detection outputs, so in other words, we're going to go to the test set and just manually tell the algorithm where the text is in each of the test examples. So in other words gonna simulate what happens if you have a text detection system with a hundred percent accuracy, for the purpose of detecting text in an image. And really the way you do that's pretty simple, right? Instead of letting your learning algorithm detect the text in the images. You wouldn't say go to the images and just manually label what is the location of the text in my test set image. And you would then let these correct or let these ground truth labels of where is the text be part of your test set. And just use these ground truth labels as what you feed in to the next stage of the pipeline, so the character segmentation pipeline. Okay? So just to say that again. By putting a checkmark over here, what I mean is I'm going to go to my test set and just give it the correct answers. Give it the correct labels for the text detection part of the pipeline. So that as if I have a perfect text detection system on my test set. What we need to do then is run this data through the rest of the pipeline. Through character segmentation and character recognition. And then use the same evaluation metric as before, to measure what was the overall accuracy of the entire system. And with perfect text detection, hopefully the performance will go up. And in this example, it goes up by 89%. And then we're gonna keep going, let's go to the next stage of the pipeline, so character segmentation. So again, I'm gonna go to my test set, and now I'm going to give it the correct text detection output and give it the correct character segmentation output. So go to the test set and manually label the correct segmentations of the text into individual characters, and see how much that helps. And let's say it goes up to 90% accuracy for the overall system. Right? So as always the accuracy of the overall system. So is whatever the final output of the character recognition system is. Whatever the final output of the overall pipeline, is going to measure the accuracy of that. And finally I'm going to build a character recognition system and give that correct labels as well, and if I do that too then no surprise I should get 100% accuracy. Now the nice thing about having done this analysis is, we can now understand what is the upside potential of improving each of these components? So we see that if we get perfect text detection, our performance went up from 72 to 89%. So that's a 17% performance gain. So this means that if we take our current system we spend a lot of time improving text detection, that means that we could

potentially improve our system's performance by 17%. It seems like it's well worth our while. Whereas in contrast, when going from text detection when we gave it perfect character segmentation, performance went up only by 1%, so that's a more sobering message. It means that no matter how much time you spend on character segmentation. Maybe the upside potential is going to be pretty small, and maybe you do not want to have a large team of engineers working on character segmentation. This sort of analysis shows that even when you give it the perfect character segmentation, your performance goes up by only one percent. That really estimates what is the ceiling, or what is an upper bound on how much you can improve the performance of your system and working on one of these components. And finally, going from character, when we get better character recognition with the forms went up by ten percent. So again you can decide is ten percent improvement, how much is worth your while? This tells you that maybe with more effort spent on the last stage of the pipeline, you can improve the performance of the systems as well. Another way of thinking about this, is that by going through these sort of analysis you're trying to think about what is the upside potential of improving each of these components. Or how much could you possibly gain if one of these components became absolutely perfect? And this really places an upper bound on the performance of that system. So the idea of ceiling analysis is pretty important, let me just answer this idea again but with a different example but more complex one.

Estimating the errors due to each component (ceiling analysis)



What part of the pipeline should you spend the most time trying to improve?

Component	Accuracy
Overall system	72% ← ↓ 17%
→ Text detection	89% ← ↓ 1%
Character segmentation	90% ← ↓ 10%
Character recognition	100%

Let's say that you want to do face recognition from images. You want to look at the picture and recognize whether or not the person in this picture is a particular friend of yours, and try to recognize the person shown in this image. This is a slightly artificial example, this isn't actually how face recognition is done in practice. But we're going to set for an example, what a pipeline might

look like to give you another example of how a ceiling analysis process might look.

Another ceiling analysis example

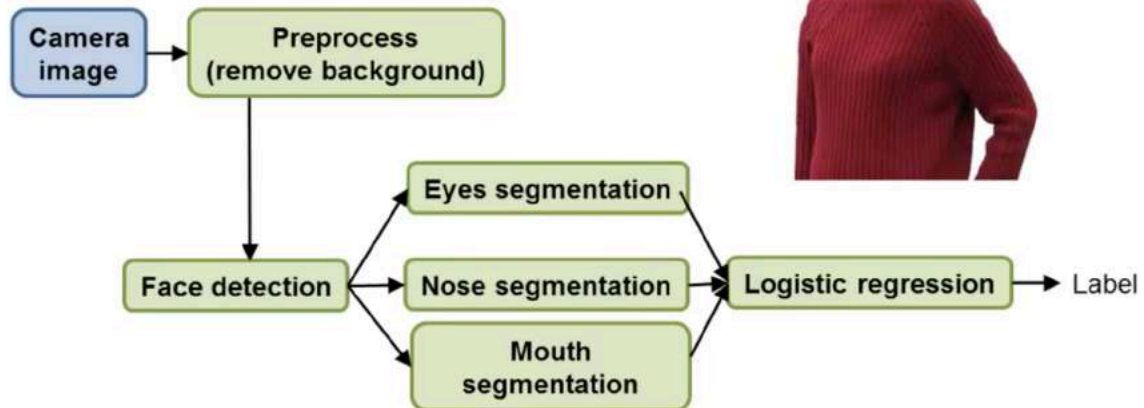
Face recognition from images
(Artificial example)



So we have a camera image, and let's say that we design a pipeline as follows, the first thing you wanna do is pre-processing of the image. So let's take this image like we have shown on the upper right, and let's say we want to remove the background. So do pre-processing and the background disappears. Next we want to say detect the face of the person, that's usually done on the learning. So we'll run a sliding Windows crossfire to draw a box around a person's face. Having detected the face, it turns out that if you want to recognize people, it turns out that the eyes is a highly useful cue. We actually are, in terms of recognizing your friends the appearance of their eyes is actually one of the most important cues that you use. So let's run another crossfire to detect the eyes of the person. So the segment of the eyes and then since this will give us useful features to recognize the person. And then other parts of the face of physical interest. Maybe segment of the nose, segment of the mouth. And then having found the eyes, the nose, and the mouth, all of these give us useful features to maybe feed into a logistic regression classifier. And there's a job with a cost priority, they'd give us the overall label, to find the label for who we think is the identity of this person. So this is a kind of complicated pipeline, it's actually probably more complicated than you should be using if you actually want to recognize people, but there's an illustrative example that's useful to think about for ceiling analysis.

Another ceiling analysis example

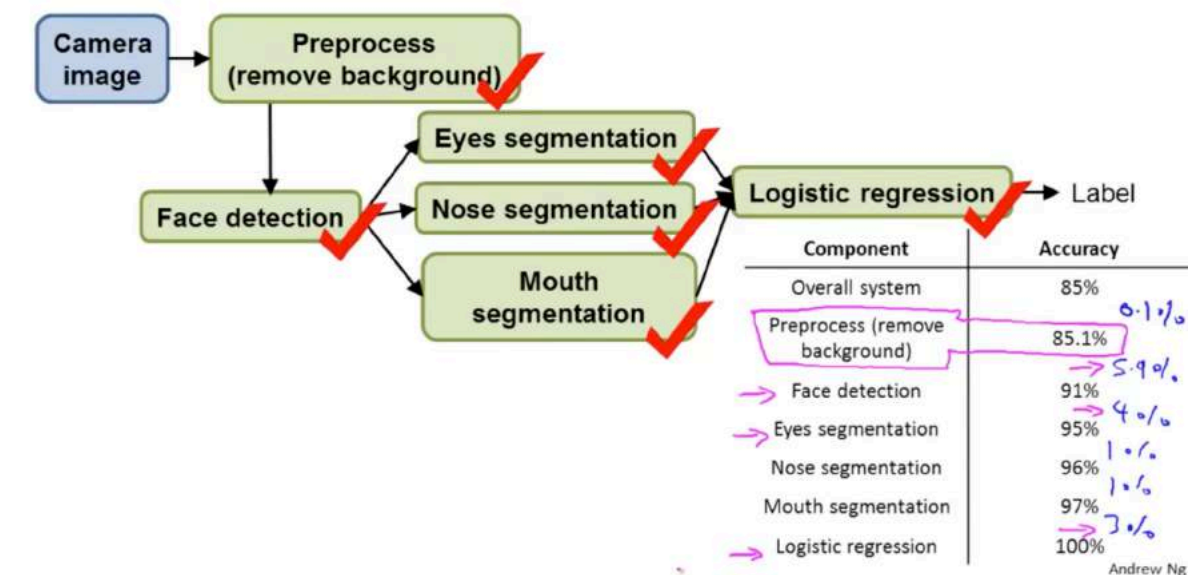
Face recognition from images
(Artificial example)



So how do you go through ceiling analysis for this pipeline. Well we step through these pieces one at a time. Let's say your overall system has 85% accuracy. The first thing I do is go to my test set and manually give it the full background segmentation. So manually go to the test set. And use Photoshop or something to just tell it where's the background and just manually remove the graph background, so this is a ground true background, and see how much the accuracy changes. In this example the accuracy goes up by 0.1%. So this is a strong sign that even if you have perfect background segmentation, the form is, even with perfect background removal the performance or your system isn't going to go up that much. So it's maybe not worth a huge effort to work on pre-processing on background removal. Then quickly goes to test set give it the correct face detection images then again step through the eyes nose and mouth segmentation in some order just pick one order. Just give the correct location of the eyes. Correct location in noses, correct location in mouth, and then finally if I just give it the correct overall label I can get 100% accuracy. And so as I go through the system and just give more and more components, the correct labels in the test set, the performance of the overall system goes up and you can look at how much the performance went up on different steps. So from giving it the perfect face detection, it looks like the overall performance of the system went up by 5.9%. So that's a pretty big jump. It means that maybe it's worth quite a bit effort on better face detection. Went up 4% there, it went up 1% there. 1% there, and 3% there. So it looks like the components that most work are while are, when I gave it perfect face detection system went up by 5.9 performance when given perfect eyes segmentation went to four percent. And then my final which is cost for well there's another three percent, gap there maybe. And so this tells maybe whether the components are most worthwhile

working on. And by the way I want to tell you a true cautionary story. The reason I put this in this preprocessing background removal is because I actually know of a true story where there was a research team that actually literally had to people spend about a year and a half, spend 18 months working on better background removal. But actually I'm obscuring the details for obvious reasons, but there was a computer vision application where there's a team of two engineers that literally spent about a year and a half working on better background removal, actually worked out really complicated algorithms and ended up publishing one research paper. But after all that work they found that it just did not make huge difference to the overall performance of the actual application they were working on and if only someone were to do ceiling analysis before hand maybe they could have realized. And one of them said to me afterward. If only you've did this sort of analysis like this maybe they could have realized before their 18 months of work. That they should have spend their effort focusing on some different component then literally spending 18 months working on background removal.

Another ceiling analysis example



Question

Suppose you perform ceiling analysis on a pipelined machine learning system, and when we plug in the ground-truth labels for one of the components, the performance of the overall system improves very little. This probably means: (check all that apply)

☐ We should dedicate significant effort to collecting more data for that component.

Un-selected is correct

☒ It is probably not worth dedicating engineering resources to improving that component of the system.

Correct

☒ If that component is a classifier training using gradient descent, it is probably not worth running gradient descent for 10x as long to see if it converges to better classifier parameters.

Correct

☐ Choosing more features for that component may help (reducing bias), and reducing the number of features for that component (reducing variance) is unlikely to do so.

Un-selected is correct

So to summarize, pipelines are pretty pervasive in complex machine learning applications. And when you're working on a big machine learning application, your time as developer is so valuable, so just don't waste your time working on something that ultimately isn't going to matter. And in this video we'll talk about this idea of ceiling analysis, which I've often found to be a very good tool for identifying the component of a video as you put focus on that component and make a big difference. Will actually have a huge effect on the overall performance of your final system. So over the years working machine learning, I've actually learned to not trust my own gut feeling about what components to work on. So very often, I've work on machine learning for a long time, but often I look at a machine learning problem, and I may have some gut feeling about oh, let's jump on that component and just spend all the time on that. But over the years, I've come to even trust my own gut feelings and learn not to trust gut feelings that much. And instead, if you have a sort of machine learning problem where it's possible to structure things and do a ceiling analysis, often there's a much better and much more reliable way for deciding where to put a focused effort, to really improve the performance of some component. And be kind of reassured that, when you do that, it won't actually have a huge effect on the final performance of the overall system.

Review

Quiz

1. Suppose you are running a sliding window detector to find text in images. Your input images are 1000x1000 pixels. You will run your sliding windows detector at two scales, 10x10 and 20x20 (i.e., you will run your classifier on lots of 10x10 patches to decide if they contain text or not; and also on lots of 20x20 patches), and you will "step" your detector by 2 pixels each time. About how many times will you end up running your classifier on a single 1000x1000 test set image?

- ☐ 100,000
- ☐ 1,000,000
- ☐ 250,000
- ☒ 500,000

2. Suppose that you just joined a product team that has been developing a machine learning application, using $m = 1,000$ training examples. You discover that you have the option of hiring additional personnel to help collect and label data. You estimate that you would have to pay each of the labellers \$10 per hour, and that each labeller can label 4 examples per minute. About how much will it cost to hire labellers to label 10,000 new training examples?

☐ \$10,000

☒ \$400

☐ \$600

☐ \$250

3. What are the benefits of performing a ceiling analysis? Check all that apply.

☐ If we have a low-performing component, the ceiling analysis can tell us if that component has a high bias problem or a high variance problem.

☒ It can help indicate that certain components of a system might not be worth a significant amount of work improving, because even if it had perfect performance its impact on the overall system may be small.

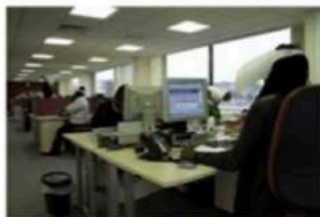
☒ It gives us information about which components, if improved, are most likely to have a significant impact on the performance of the final system.

☐ A ceiling analysis helps us to decide what is the most promising learning algorithm (e.g., logistic regression vs. a neural network vs. an SVM) to apply to a specific component of a machine learning pipeline.

4. Suppose you are building an object classifier, that takes as input an image, and recognizes that image as either containing a car ($y = 1$) or not ($y = 0$). For example, here are a positive example and a negative example:



Positive example ($y = 1$)

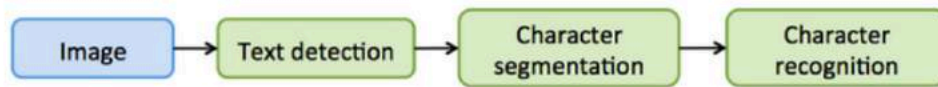


Negative example ($y = 0$)

After carefully analyzing the performance of your algorithm, you conclude that you need more positive ($y = 1$) training examples. Which of the following might be a good way to get additional positive examples?

- ☒ Mirror your training images across the vertical axis (so that a left-facing car now becomes a right-facing one).
- ☐ Take a few images from your training set, and add random, gaussian noise to every pixel.
- ☐ Take a training example and set a random subset of its pixel to 0 to generate a new example.
- ☐ Select two car images and average them to make a third example.

5. Suppose you have a PhotoOCR system, where you have the following pipeline:



You have decided to perform a ceiling analysis on this system, and find the following:

Component	Accuracy
Overall System	70%
Text Detection	72%
Character Segmentation	82%
Character Recognition	100%

Which of the following statements are true?

- ☒ There is a large gain in performance possible in improving the character recognition system.
- ☒ Performing the ceiling analysis shown here requires that we have ground-truth labels for the text detection, character segmentation and the character recognition systems.
- ☐ The least promising component to work on is the character recognition system, since it is already obtaining 100% accuracy.
- ☐ The most promising component to work on is the text detection system, since it has the lowest performance (72%) and thus the biggest potential gain.

Conclusion

Welcome to the final video of this Machine Learning class. We've been through a lot of different videos together. In this video I would like to just quickly

summarize the main topics of this course and then say a few words at the end and that will wrap up the class.

So what have we done? In this class we spent a lot of time talking about supervised learning algorithms like linear regression, logistic regression, neural networks, SVMs. for problems where you have labelled data and labelled examples like $x(i)$, $y(i)$. And we also spent quite a lot of time talking about unsupervised learning like K-means clustering, Principal Components Analysis for dimensionality reduction and Anomaly Detection algorithms for when you have only unlabelled data $x(i)$. Although Anomaly Detection can also use some labelled data to evaluate the algorithm. We also spent some time talking about special applications or special topics like Recommender Systems and large scale machine learning systems including parallelized and rapid-use systems as well as some special applications like sliding windows object classification for computer vision. And finally we also spent a lot of time talking about different aspects of, sort of, advice on building a machine learning system. And this involved both trying to understand what is it that makes a machine learning algorithm work or not work. So we talked about things like bias and variance, and how regularization can help with some variance problems. And we also spent a little bit of time talking about this question of how to decide what to work on next. So, how to prioritize how you spend your time when you're developing a machine learning system. So we talked about evaluation of learning algorithms, evaluation metrics like precision recall, F1 score as well as practical aspects of evaluation like the training, cross-validation and test sets. And we also spent a lot of time talking about debugging learning algorithms and making sure the learning algorithm is working. So we talked about diagnostics like learning curves and also talked about things like error analysis and ceiling analysis. And so all of these were different tools for helping you to decide what to do next and how to spend your valuable time when you're developing a machine learning system. And in addition to having the tools of machine learning at your disposal so knowing the tools of machine learning like supervised learning and unsupervised learning and so on, I hope that you now not only have the tools, but that you know how to apply these tools really well to build powerful machine learning systems.

Summary: Main topics

- Supervised Learning $(x^{(i)}, y^{(i)})$
 - Linear regression, logistic regression, neural networks, SVMs
- Unsupervised Learning $x^{(i)}$
 - K-means, PCA, Anomaly detection
- Special applications/special topics
 - Recommender systems, large scale machine learning.
- Advice on building a machine learning system
 - Bias/variance, regularization; deciding what to work on next: evaluation of learning algorithms, learning curves, error analysis, ceiling analysis.

So, that's it. Those were the topics of this class and if you worked all the way through this course you should now consider yourself an expert in machine learning. As you know, machine learning is a technology that's having huge impact on science, technology and industry. And you're now well qualified to use these tools of machine learning to great effect. I hope that many of you in this class will find ways to use machine learning to build cool systems and cool applications and cool products. And I hope that you find ways to use machine learning not only to make *your* life better but maybe someday to use it to make many other people's life better as well. I also wanted to let you know that this class has been great fun for me to teach. So, thank you for that. And before wrapping up, there's just one last thing I wanted to say. Which is that: It was maybe not so long ago, that I was a student myself. And even today, you know, I still try to take different courses when I have time to try to learn new things. And so I know how time-consuming it is to learn this stuff. I know that you're probably a busy person with many, many other things going on in your life. And so the fact that you still found the time or took the time to watch these videos and, you know, many of these videos just went on for hours, right? And the fact many of you took the time to go through the review questions and that many of you took the time to work through the programming exercises. And these were long and complicate programming exercises. I wanted to say thank you for that. And I know that many of you have worked hard on this class and that many of you have put a lot of time into this class, that many of you have put a lot of yourselves into this class. So I hope that you also got a lot of out this class. And I wanted to say: Thank you very much for having been a student in this class.

