MachineLearning-Lecture14

Instructor (Andrew Ng):All right. Good morning. Just a couple quick announcements before I get started. One is you should have seen Ziko's e-mail yesterday already. Several of you had asked me about – let you know [inaudible], so I wrote those up. We posted them online yesterday. The syllabus for the midterm is everything up to and including last Wednesday's lecture, so I guess [inaudible] is on the syllabus. You can take at the notes if you want. And also practice midterm had been posted on the course website, so you can take a look at that, too. The midterm will be in Terman auditorium tomorrow at 6:00 p.m. Directions were sort of included – or links to directions were included in Ziko's e-mail. And we actually at 6:00 p.m. sharp tomorrow, so do come a little bit before 6:00 p.m. to make sure you're seated by 6:00 p.m. as we'll hand out the midterms a few minutes before 6:00 p.m. and we'll start the midterm at 6:00 p.m. Okay?

Are there any questions about midterms? Any logistical things? Are you guys excited? Are you looking forward to the midterm? All right. Okay. So welcome back, and what I want to do to is talk about – is wrap up our discussion on factor analysis, and in particular what I want to do is step through parts of the derivations for EM for factor analysis because again there are a few steps in the EM derivation that are particularly tricky, and there are specific mistakes that people often make on deriving EM algorithms for algorithms like factor analysis. So I wanted to show you how to do those steps right so you can apply the same ideas to other problems as well. And then in the second half or so of this lecture, I'll talk about principal component analysis, which is a very powerful algorithm for dimensionality reduction. We'll see later what that means.

So just a recap, in a previous lecture I described a few properties of Gaussian distributions. One was that if you have a random variable – a random value vector X that can be partitioned into two portions, X1 and X2, and if X is Gaussian with mu [inaudible] and covariance sigma where mu is itself a partition vector and sigma is sort of a partition matrix that can be written like that. So I'm just writing sigma in terms of the four sub-blocks. Then you can look at the distribution of X and ask what is the marginal distribution of say X1. And the answer we said last time was that X1 – the marginal distribution of X1 is Gaussian would mean mu and covariance sigma one one, whereas sigma one one is the upper left block of that covariance matrix sigma. So this one is no surprise.

And I also wrote down the formula for computing conditional distributions, such as what is P of X1 given X2, and last time I wrote down that the distribution of X1 given X2 would also be Gaussian with parameters that I wrote as mu of one given two and sigma of one given two where mu of one given two is – let's see [inaudible] this formula. Okay? So with these formulas will be able to locate a pair of joint Gaussian random variables – X1 and X here are both vectors – and compute the marginal and conditional distributions, so P of X1 or P of X1 given X2. So when I come back and derive the E set – actually, I'll come back and use the marginal formula in a second, and then when I come back and derive from the E step in the EM algorithm for factor analysis, I'll actually be using these two formulas again.

And again, just to continue summarizing what we did last time, we said that in factor analysis our model – let's see. This is an unsupervised learning problem, and so we're given an unlabeled training set where each SI is a vector in RN as usual. We want to model the density of X, and our model for X would be that we imagine there's a latent random variable Z that's generating this [inaudible] zero mean in identity covariance. And Z will be some low dimensional thing [inaudible] and we [inaudible]. And we imagine that X is generated as mu plus lambda Z plus epsilon where epsilon is a Gaussian random variable with mean zero and a covariance matrix psi. And so the parameters of this model are mu which N-dimensional vector, lambda, which is an N by D-dimensional vector – matrix, and psi which is N by N and is diagonal. N is a diagonal matrix. So the cartoon I drew last time for factor analysis was – I said that maybe that's the typical example of data point ZI if – and in this example I had D equals one, N equals two. So Z in this example is one-dimensional. D equals one. And so you take this data, map it to say [inaudible] mu plus lambda Z and that may give you some set of points there.

And lastly, this model was envisioning that you'd place a little Gaussian bump around each of these say and sample – and the Xs are then maybe – that would be a typical sample of the Xs under this model. So how [inaudible] the parameters of this model? Well, the joint distribution of Z and X is actually Gaussian where parameters given by some vector [inaudible] mu XZ, and sum covariance matrix sigma. And [inaudible] what those two things are, this vector [inaudible] is a vector of zeroes appended to the vector mu. And the matrix sigma is this partitioned matrix.

We also worked this out last time. So you can ask what is the distribution of X under this model, and the answer is under this model X is Gaussian with mean mu and covariance lambda lambda [inaudible] plus psi. So let's just take the second block of the mean vector and take that lower right hand corner block for the covariance matrix, and so this is really my formula for computing the marginal distribution of a Gaussian, except that I'm computing the marginal distribution of the second half of the vector rather than the first half. So this is the marginal distribution of X under my model. And so if you want to learn –

Student:[Inaudible] initial distribution [inaudible]?

Instructor (Andrew Ng):Let's see. Oh, yes. Yes, so in this one I'm breaking down the – this is really I'm specifying the conditional distribution of X given Z. So the conditional distribution of X given Z – this is Gaussian – would mean mu plus lambda Z and covariance psi. This is what that [inaudible]. So since this is the marginal distribution of X, given my training set of M unlabeled examples, I can actually write down the log likelihood of my training set. So the log likelihood of my training set – actually, no. Let's just write down the likelihood. So the likelihood of my parameters given my training set is the product from I equals one to M of P of XI given the parameters.

I can actually write down what that is because XI is Gaussian with parameter mu and variance lambda lambda transpose times psi, so you can actually write this down as one over two pi to the N over two determined of – and then times E to the minus one half X of – so that's my formula for the density of a Gaussian that has mean mu and covariance lambda lambda transpose plus psi. So this is my likelihood of the parameters given a training set. And one thing you could do is actually take this likelihood and try to maximize it in terms of the parameters, try to find the [inaudible] the parameters. But if you do that you find that – if you sort of try – take [inaudible] to get the law of likelihood, take derivatives, set derivatives equal to zero, you find that you'll be unable to solve for the maximum of this analytically. You won't be able to solve this [inaudible] likelihood estimation problem.

If you take the derivative of this with respect to the parameters, set the derivatives equal to zero and try to solve for the value of the parameters lambda, mu, and psi [inaudible] so you won't be able to do that [inaudible]. So what we'll use instead to estimate the parameters in a factor analysis model will be the EM algorithm.

Student: Why is the law of likelihood P of X and not P of X [inaudible] X and Z or something?

Instructor (Andrew Ng):Oh, right. So the question is why is the likelihood P of X and not P of X given Z or P of X and Z. The answer is let's see – we're in the – so by analogy to the mixture of Gaussian distributions models, we're given a training set that just comprises a set of unlabeled training examples that – for convenience for whatever reason, it's easier for me to write down a model that defines the joint distribution on P of X comma Z.

But what I would like to do is really maximize – as a function of my parameters – I'm using theta as a shorthand to denote all the parameters of the model – I want to maximize the probability of the data I actually observe. And this would be actually [inaudible] theta [inaudible] from I equals one to M, and this is really integrating out Z.

So I only ever get to observe the Xs and the Zs are latent random variables or hidden random variables. And so what I'd like to do is do this maximization in which I've implicitly integrated out Z. Does that make sense? Actually, are there other questions about the factor analysis models? Okay. So here's the EM algorithm. In the E step, we compute the conditional distribution of ZI given XI in our current setting of the parameters. And in the M step, we perform this maximization. And if this looks a little bit different from the previous versions of EM that you've seen, the only difference is that now I'm integrating over ZI because ZI is now a Gaussian random variable, is now this continuous value thing, so rather than summing over ZI, I now integrate over ZI. And if you replace [inaudible] of a sum, you [inaudible] ZI then this is exactly the M step you saw when we worked it out from the mixture of Gaussian [inaudible].

So it turns out that in order to implement the E step and the M step, there are just two little things – there are actually sort of three key things that are different from the models that you saw previously, so I wanna talk about them. The first one is that the first of the [inaudible] which is that in the E step, Z is now a continuous value random variable, so you now need a way to represent these continuous value densities, probably density functions to represent QI of Z. So fortunately in this probably it's not difficult to do, and in particular the conditional distribution of ZI given XI and our parameters which I'm gonna omit from this equation is going to be Gaussian with mean and covariance given by these two things, so I write like that, where this is going to be equal to the vector zero minus –

And the way I got this formula was – if you match this to the formula I had previously for computing conditional distributions of Gaussians, this corresponded to the terms mu one minus sigma one two sigma two two inverse times two minus mu. So those are the terms corresponding to the form I had previously for computing the marginal distributions of a Gaussian. And that is going to be given by that, and again those are the terms corresponding to the formulas I had for the very first thing I did, the formulas for computing conditional distributions of Gaussians. And so this is E step. You need to compute the Q distribution. You need to compute QI of ZI, and to do that what you actually do is you compute this vector mu of ZI given XI and sigma of ZI given XI, and together these represent the mean and covariance of the distribution Q where Q is going to be Gaussian, so that's the E step. Now here's the M step then. I'll just mention there are sort of two ways to derive M steps for especially Gaussian models like these. Here's the key trick I guess which is that when you compute the M step, you often need to compute integrals that look like these. And then there'll be some function of ZI. Let me just write ZI there, for instance.

So there are two ways to compute integrals like these. One is if you write out – this is a commonly made mistake in – well, not mistake, commonly made unnecessary complication I guess. One way to try to compute this integral is you can write this out as integral over ZI. And while we know what QI is, right? QI is a Gaussian so two pi, so D over two – the covariance of QI is this sigma given XI which you've computed in the E step, and then times E to the minus one half ZI minus mu of ZI given XI transpose sigma inverse – so that's my Gaussian density. And then times ZI ZZI. And so writing those out is the unnecessarily complicated way to do it because once you've written out this integral, if you want to actually integrate – that's the times, multiplication – if you actually want to evaluate this integral, it just looks horribly complicated. I'm not quite sure how to evaluate this. By far, the simpler to evaluate this integral is to recognize that this is just the expectation with respect to ZI drawn from the distribution QI of the random variable ZI. And once you've actually got this way, you notice that this is just the expected value of ZI under the distribution QI, but QI is a Gaussian distribution with mean given by that mu vector and covariance given by that sigma vector, and so the expected value of ZI under this distribution is just mu of ZI given XI. Does that make sense? So by making those observations that this is just an expected value, there's a much easier way to compute that integral. [Inaudible]. So we'll apply the same idea to the M step. So the M step we want to maximize this. There's also sum over I – there's a summation over I outside, but this is essentially the term inside the arg max of the M step where taking the integral of a ZI and just observe that that's actually – I guess just form some expectation respect to the random variable ZI of this thing inside. And so this simplifies to the following.

And it turns out actually all I did here was use the fact that P of X given Z times P of Z equals P over X comma Z. Right? That's just combining this term and that term gives you the numerator in the original. And so in turns out that for factor analysis in these two terms, this is the only one that depends on the parameters. The distribution P of ZI has no parameters because ZI was just drawn from a Gaussian with zero mean and identity covariance. QI of Z was this fixed Gaussian. It doesn't depend on the parameters theta, and so in the M step we really just need to maximize this one term with respect to all parameters, mu, lambda, and psi. So let's see. There's sort of one more key step I want to show but to get there I have to write down an unfortunately large amount of math, and let's go into it. Okay.

So in the M step, we want to maximize and all expectations with respect to ZI drawn from the distributions QI, and sometimes I'll be sloppy and just omit this. And now this distribution P of XI given ZI, that is a Gaussian density because XI given ZI – this is Gaussian with mean given by mu plus lambda Z and covariance psi. And so in this step of the derivation, I will actually go ahead and substitute in the formula for Gaussian density. So I will go ahead and take those and plug it into here, one over two pi to the N over two, psi inverse E to the dot, dot, dot. So I will go ahead and plug in the Gaussian density. And when you do that, you find that you get expectation – excuse me. I forgot to say to maintain – to not make the derivation too complicated, I'm actually just going to maximize this with respect to the parameters lambda. So let me just show how the – so you want to maximize this with respect to lambda, psi, and mu, but just to keep the amount of math I'm gonna do in class sane, I'm just going to show how to maximize this with respect to the matrix lambda, and I'll pretend that psi and mu are fixed. And so if you substitute in the Gaussian density, you get some expected value of the constant. The constant may depend on psi but not on lambda. Then mine is this thing. And this quadratic term essentially came from the exponent in your Gaussian density. When I take log of exponent, then you end up with this quadratic term.

And so if you take the derivatives of the expression above with respect to the matrix lambda and you set that to zero – we want to maximize this expression with respect to the parameters lambda, so you take the derivative of this with respect to lambda – excuse me. That's a minus sign – and set the derivate of this expression to zero because you set derivatives to zero to maximize things, right? When you do that and simplify, you end up with the following. And so that's the – in the M step, this is the value you should get that you use to update your parameters lambda. And again, the expectations are with respect to ZI drawn from the distributions QI. So the very last step of this derivation is we need to work out what these two expectations are. And so the very first term EZI transpose I guess is just mu of ZI give XI transpose because the QI distribution has mean given by [inaudible]. To work out the other term, let me just remind you that if you have a random variable Z that's Gaussian with mean mu and covariance sigma, then the covariance sigma is EZZ transpose minus EZ EZ transpose. That's one of the definitions of the covariance, and so this implies that EZZ transpose equals sigma plus EZ EZ transpose. And so this second term here becomes sigma ZI to the mu I plus – given XI. Okay?

And so that’s how you compute E of ZI transpose and E of ZI ZI transpose and substitute them back into this formula. And you would then have your M step update to the parameter matrix lambda. And the last thing I want to point out in this derivation is that it turns out – it’s probably because of the name EM algorithm, expectation maximization, one common mistake for the EM algorithm is in the E step, some want to take the expectation of the random variable Z, and then in the M step they just plug in the expected value everywhere you see it. So in particular, one common mistake in deriving EM for factor analysis is to look at this and say, “Oh, look. I see ZZ transpose. Let’s just plug in the expected value under the Q distribution.” And so plug in that – mu of ZI given XI times mu of ZI given XI transpose – into that expectation, and that would be an incorrect derivation of EM because it’s missing this other term, sigma of ZI given XI. So one common misconception for EM is that in the E step you just compute the expected value of the hidden random variable, and the M step you plug in the expected value. It turns out in some algorithms that turns out to be the right thing to do. In the mixture of Gaussians and the mixture of [inaudible] models, that would actually give you the right answer, but in general the EM algorithm is more complicated than just taking the expected values of the random variables and then pretending that they were sort of observed at the expected values.

So I wanna go through this just to illustrate that step as well. So just to summarize the three key things to keep in – that came up in this variation were, 1.) That for the E step, we had a continuous Gaussian random variable, and so to compute the E step, we actually compute the mean and covariance of the distribution QI. The second thing that came up was in the M step when you see these integrals, sometimes if you interpret that as expectation then the rest of the math becomes much easier. And the final thing was again in the M step, the EM algorithm is derived by a certain maximization problem that we solve. It is not necessarily just plugging the expected value of ZI everywhere.

Let’s see. I feel like I just did a ton of math and wrote down way too many equations. And even doing this, I was skipping many steps. So you can go to the lecture notes to see all the derivations of the steps I skipped, like how you actually take derivatives with respect to the matrix lambda, and how to compute the updates for the other parameters as well, for mu and for psi, because this is only for lambda. And so that’s the factor analysis algorithm. Justin?

Student:I was just wondering in the step in the lower right board, you said that the second term doesn’t have any parameters that we’re interested in. The first term has all the parameters, so we’ll [inaudible], but it seems to me that QI has a lot of parameters [inaudible].

Instructor (Andrew Ng):I see. Right. Let’s see. So the question was doesn’t the term QI have parameters? So in the EM algorithm QI is – it actually turns out in the EM algorithm, sometimes P of ZI may have parameters, but QI of ZI may never have any parameters. In the specific case of factor analysis, P of ZI doesn’t have parameters. In other examples, the mixture of Gaussian models say, ZI was a multinomial random variable, and so in that example PI of ZI has parameters, but it turns out that Q of ZI will never have any parameters.

And in particular, QI of ZI is going to be – is this Gaussian distribution – is Gaussian with mean given by mu of ZI given XI and covariance sigma ZI given XI. And so it's true that mu and sigma may themselves have depended on the values of the parameters I had in the previous iteration of EM, but the way to think about Q is I'm going to take the parameters from the previous iteration of the algorithm and use that to compute what QI of ZI is. And that's the E second EM algorithm. And then once I've computed what QI of Z is, then this is a fixed distribution. I'm gonna use these fixed values for mu and sigma, and just keep these two values fixed as I run the M step.

Student: So that's – I guess I was confused because in the second point over there's a lot of – it looks like they're parameters, but I guess they're old iterations of the parameters.

Instructor (Andrew Ng):Oh, yeah. Yes, you're right. When I wrote down QI of ZI that was a function of – so yeah – the parameters from the previous iteration. And I want to compute the new set of parameters. Okay. More questions? So this is probably the most math I'll ever do in a lecture in this entire course. Let's now talk about a different algorithm. Actually, which board was I on? So what I want to do now is talk about an algorithm called principal components analysis, which is often abbreviated PCA. Here's the idea. PCA has a very similar idea as factor analysis, but it sort of maybe gets to the problem a little more directly than just factor analysis.

So the question is given – so we're still doing unsupervised learning, so given a training set of M examples where each XI is an N-dimensional vector as usual, what I like to do is reduce it to a lower dimensional data set where K is strictly less than N, and quite often will be much smaller than N. So I'll give a couple examples of why we want to do this. Imagine that you're given a data set that contains measurements and unknown to you – measurements of, I don't know, people or something – and unknown to you, whoever collected this data actually included the height of the person in centimeters as well as the height of the person in inches. So because of rounding off to the nearest centimeter or rounding off to the nearest inch the values won't exactly match up, but along two dimensions of this data anyway, it'll lie extremely close to a straight line, but it won't lie exactly on a straight line because of rounding off to the nearest centimeter or inch, but lie very close to the straight line.

And so we have a data set like this. It seems that what you really care about is that axis, and this axis is really the variable of interest. That's maybe the closest thing you have to the true height of a person. And this other axis is just noise. So if you can reduce the dimension of this data from two-dimensional to one-dimensional, then maybe you can get rid of some of the noise in this data. Quite often, you won't know that this was cm and this was inches. You may have a data set with a hundred attributes, but you just didn't happen to notice one was cm and one was inches. Another example that I sometimes think about is some of you know that my students and I work with [inaudible] helicopters a lot. So we imagined that you take surveys of quizzes, measurements of radio control helicopter pilots. Maybe one axis you have measurements of your pilot's skill, how skillful your helicopter pilot is, and on another axis maybe you measure how much they actually enjoy flying. And maybe – this is really – maybe this is actually roughly one-dimensional data, and there's some variable of interest, which I'll call maybe pilot attitude that somehow determines their skill and how much they enjoy flying. And so again, if you can reduce this data from two dimensions to one-dimensional, maybe you'd have a slightly better of measure of what I'm calling loosely pilot attitude, which may be what you really wanted to [inaudible]. So let's talk about an algorithm to do this, and I should come back and talk about more applications of PCA later.

So here's the algorithm. Before running PCA normally you will preprocess the data as follows. I'll just write this out, I guess. I know some of you are writing. So this is maybe just an unnecessary amount of writing to say that compute a mean of my training sets and subtract out the means, so now I've zeroed out the mean of my training sets. And the other step is I'll compute the variance of each of features after zeroing out the mean, and then I'll divide each feature by the standard deviation so that each of my features now has equal variance. So these are some standard preprocessing steps we'll often do for PCA.

I'll just mention that sometimes the second step is usually done only when your different features are on different scales. So if you're taking measurements of people, one feature may be the height, and another may be the weight, and another may be the strength, and another may be how they age or whatever, all of these quantities are on very different scales, so you'd normally normalize the variance. Sometimes if all the XIs are the same type of thing – so for example if you're working with images and the XIJs are the pixels that they're – are on the same scale because they're all pixel intensity values ranging from zero to 255, then you may omit this step. So after preprocessing, let's talk about how we would find the main axes along which the data varies. How would you find a principal axes of variations [inaudible]? So to do that let me describe – let me just go through one specific example, and then we'll formalize the algorithm. Here's my training set comprising five examples. It has roughly zero mean. And the variance under X1 and X2 axes are the same. There's X1, there's X2 axes. And so the principal axes and variation of this data is roughly the positive 45-degree axis, so what I'd like to do is have my algorithm conclude that this direction that I've drawn with this arrow U is the best direction onto which to project the data, so the axis by which my data really varies.

So let's think about how we formalize. One thing we want to look at is suppose I take that axis – this is really the axis onto which I want to project – that I want to use to capture most of the variation of my data. And then when I take my training set and project it onto this line, then I get that set of points. And so you notice that those dots, the projections of my training sets onto this axis, has very large variance. In contrast, if I were to choose a different direction – let's say this is really [inaudible] the worst direction onto which to project my data. If I project all my data onto this axis, then I find that the projects of my data onto the purple line, onto this other line has much smaller variance, that my points are clustered together much more tightly. So one way to formalize this notion of finding the main axis of variations of data is I would like to find the vector U, I would like to find the direction U so that when I project my data onto that direction, the projected points vary as much as possible. So in other words, I'm gonna find a direction so that when I project my data onto it, the projections are largely widely spaced out. There's large variance.

So let's see. So I want to find the direction U. Just as a reminder, if I have a vector U of norm one, then the length of our vector XI projected – then the vector XI projected onto U has length XI transpose U. To project a vector onto – to project X onto unit vector, the length of the projection's just XI transpose U. And so to formalize my PCA problem, I'm going to choose a vector U to maximize – so I choose U and this is subject to the constraint that the norm of U, that the length of U is one – but I'm going to maximize – let's see, sum from I equals one to M – the length of the projection of the vectors X onto U. In particular, I want the sum of square distances of the projections to be far from the origin. I want the projections of X onto U to have large variance.

And just to simplify some of the math later, let me put a one over M in front. And so that quantity on the right is equal to one over M. Let's see. U transpose XI times XI transpose U, and so can simplify and I get – this is U transpose times [inaudible] U. So I want to maximize U transpose times some matrix times U subject to the constraint that the length of U must be equal to one. And so some of you will recognize that this means U must be the principal eigenvector of this matrix in the middle. So let me just write that down and say a few more words about it.

So this implies that U is the principal eigenvector of the matrix, and I'll just call this matrix sigma. It actually turns out to be a covariance matrix, [inaudible] equals one to M transpose. Actually, let's check. How many of you are familiar with eigenvectors? Oh, cool. Lots of you, almost all of you. Great. What I'm about to say is very extremely familiar, but it's worth saying anyway. So if you have a matrix A and a vector U, and they satisfy AU equals lambda U, then this is what it means for U to be an eigenvector of the matrix A. And the value lambda here is called an eigenvalue. And so the principal eigenvector is just the eigenvector that corresponds to the largest eigenvalue. One thing that some of you may have seen, but I'm not sure that you have, and I just wanna relate this to stuff that you already know as well, is that optimization problem is really maximize U transpose U subject to the norm of U is equal to one and – let me write that constraint as that U transpose U equals one.

And so to solve this constrained optimization problem, you write down the Lagrangian [inaudible] lambda, where that's the Lagrange multiplier because there's a constraint optimization. And so to actually solve this optimization, you take the derivative of L with respect to U and that gives you sigma U minus lambda U. You set the derivative equal to zero and this shows that sigma U equals lambda U, and therefore the value of U that solves this constraint optimization problem that we care about must be an eigenvector of sigma. And in particular it turns out to be the principal eigenvector. So just to summarize, what have we done? We've shown that given a training set, if you want to find the principal axis of a variation of data, you want to find the 1-D axis on which the data really varies the most, what we do is we construct the covariance matrix sigma, the matrix sigma that I wrote down just now, and then you would find the principal eigenvector of the matrix sigma. And this gives you the best 1-D subspace onto which to project the data.

And more generally, you would choose – if you wanna K-dimensional subspace onto which project your date, you would then choose U1 through UK to be the top K eigenvectors of sigma. And by top K eigenvectors, I just mean the eigenvectors corresponding to the K highest eigenvalues. I guess I showed this only for the case of a 1-D subspace, but this holds more generally. And now the eigenvectors U1 through UK represent – give you a new basis with which to represent your data.

Student:[Inaudible] diagonal?

Instructor (Andrew Ng):Let's see. So by convention, the PCA will choose orthogonal axes. So I think this is what I'm saying. Here's one more example. Imagine that you're a three-dimensional set, and imagine that your three-dimensional data set – it's very hard to draw in 3-D on the board, so just let me try this. Imagine that here my X1 and X2 axes lie on the plane of the board, and the X3 axis points directly out of the board. Imagine that you have a data set where most of the data lies on the plane of the board, but there's just a little bit of fuzz. So imagine that the X3 axis points orthogonally out of the board, and all the data lies roughly in the plane of the board, but there's just a tiny little bit of fuzz so that some of the data lies just a couple of millimeters off the board.

So you run a PCA on this data. You find that U1 and U2 will be some pair of bases that essentially lie in the plane of the board, and U3 will be an orthogonal axis at points roughly out of the plane of the board. So if I reduce this data to two dimensions, then the bases U1 and U2 now give me a new basis with which to construct my lower dimensional representation of the data.

Just to be complete about what that means, previously we have say fairly high dimensional input data XI and RN, and now if I want to represent my data in this new basis given by U1 up to UK, instead I would take each of my original training examples XI and I would now represent it or replace it with a different vector which I call YI, and that would be computed as U1 transpose XI U2 transpose XI. And so the YIs are going to be K-dimensional where K will be less – your choice of K will be less than N, so this represents a lower dimensional representation of your data, and serves as an approximate representation of your original data where you're using only K numbers to represent each training example rather than N numbers. Let's see.

Student:[Inaudible] to have eigenvectors [inaudible] trivial eigenvectors?

Instructor (Andrew Ng):Is it possible not to have eigenvectors but have trivial eigenvectors?

Student:[Inaudible] determined by [inaudible]? Is it a condition wherein [inaudible].

Instructor (Andrew Ng):Let's see. What do you mean by trivial eigenvectors?

Student:[Inaudible] linear algebra from [inaudible].

Instructor (Andrew Ng):Oh, okay. Yes, so I see. Let me see if I can get this right. So there are some matrices that are – I think the term is degenerate – that don't have a full set of eigenvectors. Sorry, deficient I think is what it's called. Is that right, Ziko? Yeah. So some matrices are deficient and actually don't have a full set of eigenvectors, like an N by N matrix that does not have N distinct eigenvectors, but that turns out not to be possible in this case because the covariance matrix sigma is symmetric, and symmetric matrices are never deficient, so the matrix sigma will always have a full set of eigenvectors.

It's possible that if you – there's one other issue which is repeated eigenvalues. So for example, it turns out that in this example if my covariance matrix looks like this, it turns out that the identity of the first two eigenvectors is ambiguous. And by that I mean you can choose this to be U1 and this to be U2 or I can just as well rotate these eigenvectors and choose that to be U1 and that to be U2, or even choose that to be U1 and that to be U2, and so on.

And so when you apply PCA, one thing to keep in mind is sometimes eigenvectors can rotate freely within their subspaces when you have repeated eigenvalues or close to repeated eigenvalues. And so the way to think about the vectors U is think of as a basis with which to represent your data, but the basis vectors can sometimes rotate freely, and so it's not always useful to look at the eigenvectors one at a time, and say this is my first eigenvector capturing whatever, the height of a person, or this is my second eigenvector, and it captures their skill at [inaudible] or whatever. That's a very dangerous thing to do when you do PCA. What is meaningful is the subspace spanned by the eigenvectors, but looking at the eigenvectors one at a time is sometimes a dangerous thing to do because they can often freely. Tiny numerical changes can cause eigenvectors to change a lot, but the subspace spanned by the top K eigenvectors will usually be about the same.

It actually turns out there are multiple possible interpretations of PCA. I'll just give one more without proof, which is that [inaudible] whatever – given a training set like this, another view of PCA is – and let me just choose a direction. This is not the principal components. I choose some direction and project my data onto it. This is clearly not the direction PCA would choose, but what you can think of PCA as doing is choose a subspace onto which to project your data so as to minimize the sum of squares differences between the projections and the [inaudible] points. So in other words, another way to think of PCA is trying to minimize the sum of squares of these distances between the dots, the points X and the dots onto which I'm projecting the data. It turns out they're actually – I don't know. There's sort of nine or ten different interpretations of PCA. This is another one. There are a bunch of ways to derive PCA. You get play with some PCA ideas more in the next problem set.

What I want to do next is talk about a few applications of PCA. Here are some ways that PCA is used. One is visualization. Very often you have high dimensional data sets. Someone gives you a 50-dimensional data set, and it's very hard for you to look at a data set that's 50-dimensional and understand what's going on because you can't plot something in 50 dimensions. So common practice if you want to visualize a very high dimensional data set is to take your data and project it into say a 2-D plot, or project it into a 3-D plot, so you can render like a 3-D display on a computer so you can better visualize the data and look for structure. One particular example that I learned about of doing this recently was in Krishna Shenoy's lab here in Stanford in which he had readings from 50 different parts of a monkey brain. I actually don't know it was the number 50. It was tens of different parts of the monkey brain, and so you'd have these 50-dimensional readings, 50-dimensional vectors correspond to different amounts of electrical activity in different parts of the monkey brain. It was actually 50 neurons, but tens of neurons, but it was tens of neurons in the monkey brain, and so you have a 50-dimensional time series, and it's very hard to visualize very high dimensional data.

But what he understood is that – was use PCA to project this 50-dimensional data down to three dimensions, and then you can visualize this data in three dimensions just using 3-D plot, so you could visualize what the monkey is thinking over time. Another common application of PCA is compression, so if you have high dimensional data and you want to store it with [inaudible] numbers, clearly PCA is a great way to do this. It turns out also that sometimes in machine learning, sometimes you're just given extremely high dimensional input data, and for computational reasons you don't want to deal with such high dimensional data. And so fairly – one common use of PCA is taking very high dimensional data and represent it with low dimensional subspace – it's that YI is the representation I wrote down just now – so that you can work with much lower dimensional data. And it turns out to be it's this sort of – just seems to make a fact of life that when you're given extremely high dimensional data almost all the time just in practice – of all the high dimensional data sets I've ever seen, very high dimensional data sets often have all their points lying on much lower dimensional subspaces, so very often you can dramatically reduce the dimension of your data and really be throwing too much of the information away. So let's see. If you're learning algorithms, it takes a long time to run very high dimensional data. You can often use PCA to compress the data to lower dimensions so that your learning algorithm runs much faster. And you can often do this with sacrificing almost no performance in your learning algorithm.

There's one other use of PCA for learning which is – you remember when we talked about learning theory, we said that the more features you have, the more complex your hypothesis class, if say you're doing linear classification. If you have more features, you have lots of features, then you may be more prone to overfitting. One other thing you could do with PCA is just use that to reduce the dimension of your data so that you have fewer features and you may be slightly less prone to overfitting. This particular application of PCA to learning I should say, it sometimes works. It often works. I'll actually later show a couple of examples where sort of do this and it works. But this particular application of PCA, I find looking in the industry, it also seems to be a little bit overused, and in particular – actually, let me say more about that later. We'll come back to that later. There's a couple of other applications to talk about. One is outlier detection or anomaly detection. The idea is suppose I give you a data set. You may then run PCA to find roughly the subspace on which your data lies. And then if you want to find anomalies in future examples, you then just look at your future examples and see if the lie very far from your subspace.

This isn’t a fantastic anomaly detection algorithm. It’s not a very good – the idea is if I give you a data set, you may find a little dimensional subspace on which it lies, and then if you ever find a point that’s far from your subspace, you can factor it as an anomaly. So this really isn’t the best anomaly detection algorithm, but sometimes this is done. And the last application that I want to go into a little bit more in detail is matching or to find better distance calculations. So let me say what I mean by this. I’ll go into much more detail on this last one. So here’s the idea. Let’s say you want to do face recognition [inaudible]. So let’s say you want to do face recognition and you have 100 of 100 images. So a picture of a face is like whatever, some array of pixels, and the pixels have different grayscale values, and dependent on the different grayscale values, you get different pictures of people’s faces. And so [inaudible] you have 100 of 100 pixel images, and you think of each face as a 10,000-dimensional vector, which is very high dimensional. [Inaudible] cartoon to keep in mind, and you think of here’s my plot of my 100-dimensional space, and if you look at a lot of different pictures of faces, each face will be along what will be some point in this 100,000-dimensional space. And in this cartoon, I want you to think of it as – I mean most of this data lies on a relatively low dimensional subspace because in this 10,000-dimensional space, really not all points correspond to valid images of faces. The vast majority of values, the vast majority of 10,000-dimensional images just correspond to random noise like looking things and don’t correspond to a valid image of a face. But instead the space of possible face of images probably spans a much lower dimensional subspace. [Crosstalk]

Instructor (Andrew Ng):And so what we’ll do is we’ll use a low dimensional subspace on which the data lies, and in practice a PCA dimension of 50 would be fairly typical. So when you think of – there’s some axes that really measure the shape or the appearance of the face, and there are some other axes that we’re not interested in that are maybe just random noise.

So what we’ll do is for face recognition I might give you a picture of a face and ask you what face looks the most similar to this. I’ll give you a picture of someone and ask you can you find other pictures of the same person. And so the key step to do that is to look at two faces and to compare how similar these two faces are. So here’s how we’ll use PCA. Given a face there and a different face there, the way I’ll measure the difference between these two faces is I won’t just measure the Euclidian distance similarity. Instead, I’ll take the space and project it onto my 50-dimensional subspace – take this and project it onto my 50-dimensional subspace and measure the similarity between those two points on the subspace. And so when I do that, I may end up with a face here and a face here that look very far upon the original space, but when I project them onto the subspace, they may end up looking much more similar.

So what I want to show you a second on the laptop is given each of these training examples is a 10,000-dimensional vector, I can plot that as a grayscale image and I’d get like a picture of some person’s face. What I’ll also show you on the laptop will be plots of my eigenvectors. And so the eigenvectors also live in a 10,000-dimensional subspace. And I plot that.

You get some image that's – and these images are called eigenfaces. And what PCA is doing is essentially using linear combinations of these eigenface images to approximate these XIs, and it's the eigenfaces that span – so that these bases U1, U2 and so on that span hopefully the subspace along which my faces live. [Crosstalk]

Instructor (Andrew Ng):So here's a training set comprising aligned images like these, a much larger set than is shown here, and so when you run PCA, these are some of the plots of the UIs. Remember when I said plot the eigenvectors UI in the same way that it's plotting the training examples, and so you end up with the eigenvectors being grayscale images like these, and what approximate images of faces with linear combinations of these. So I said it's dangerous to look at individual eigenvectors and you really shouldn't do that, but let me just do that a little bit anyway. So if you look at the first eigenvector, this corresponds roughly to whether the face is illuminated from the left or the right. So depending on how heavy the weight is on this image, that roughly captures variation in illumination. The second image is hard to tell. It seems to be capturing variation in overall brightness of the face. The third eigenvector capturing roughly how much of a shadow or maybe a beard or something a person has and so forth. It's dangerous to look at individual eigenvectors, but it sort of slightly informs the look of it.

Here's an example of a specific application of eigenfaces. This is from Sandy Pentland – Alex Pentland's lab at MIT. In this display, this upper leftmost image is the input to the eigenface's algorithm. The algorithm is then asked to go through a database and find the most similar faces. This second image here is what it thinks is the most similar face of the input. The next one over is the second most similar. The next one over is the third most similar and so on. And so using eigenfaces, and by measuring differences between faces in that lower dimensional subspace, it is able to do a reasonable job identifying pictures of the same person even with faces of the person removed. And the next row shows the next most similar faces and so on, so this one is the fourth most similar face and so on. This is a usual application of eigenfaces.

The last thing I wanna say is just when people tell me about machine learning problems, I do often recommend they try PCA for different things, so PCA is a useful thing to know to use for compression, visualization, and so on. But in industry, I just tend to see PCA used slightly more often. There are also some times where you see people using PCA when they really shouldn't. So just one final piece of advice for use of PCA is before you use it, also think about whether you could just do it with the original training data XI without compressing it, since I've also definitely seen people compress the data when they really didn't need to. But having said that, I also often advise people to use PCA for various problems and it often works. Okay. Sorry about running late. So let's wrap up for today.

[End of Audio]

Duration: 82 minutes