MachineLearning-Lecture15

Instructor (Andrew Ng):All right. Good morning. Welcome back. So a couple quick announcements before I go into today's technical material. So unfortunately, a long time ago, I agreed to give a few [inaudible] in Japan, so I won't be here on Wednesday. So the next lecture will be, I shall be taping it ahead of time. In particular, instead of a lecture on Wednesday, I'll be giving Wednesday's lecture tomorrow instead at 1:30 p.m. in the Skeeling 191 auditorium.

So those here, you come to class at the usual time, the taped lecture will be taped ahead of time, but then shown in this room at this time. But if any of you are free tomorrow at 1:30 p.m., I should be grateful if you can come to that, alternative of the taping of the lecture, which will be at 1:30 p.m. Skeeling 191.

I'll also send an email out with the details, so you don't need to write it down. But if any of you can make it to that, it'll make it much easier and more fun for me to teach to a live audience rather than to teach on tape to an empty room. So I'll send an email about that. We'll do this for this Wednesday's lecture as well as for the lecture on the Wednesday after the Thanksgiving break.

In case you don't want to deal with it, and this is the only time you make it, you can just show up at the usual time, and the lecture will be presented here in this room as well, and online, of course. Let's see. For the same reason, my next two office hours, which are usually on Fridays will also be moved to the following Monday because I'll be traveling. So if you're planning to come to my office hours either this week or the week after Thanksgiving, just take a look at the course website for the revised time.

In case you want to meet with me, but can't make the new time on the course website, you can also send me an email. We'll try to set up a time outside that. Okay?

But again, if you're free at 1:30 p.m. tomorrow, either those watching this online now or people in this classroom, I would be grateful if you can come to Skeeling 191. If there are any of you that couldn't ever make this regular class time and so always watch it online, you can come see a live lecture, for once, those of you watching online.

Are there any administrative questions about that? Okay? Let's go into today's technical material.

Welcome back. What I want to do today is continue a discussion of principal components analysis, or PCA. In particular, there's one more application that I didn't get to in the last lecture on [inaudible] indexing, LSI. Then I want to spend just a little time talking about how to implement PCA, especially for very large problems. In particular, I'll spend just a little bit of time talking about singular value decomposition, or the SVD implementation of principal component analysis.

So the second half of today's lecture, I want to talk about the different algorithm called independent component analysis, which is, in some ways, related to PCA, but in many other ways, it also manages to accomplish very different things than PCA. So with this lecture, this will actually wrap up our discussion on unsupervised learning. The next lecture, we'll start to talk about reinforcement learning algorithms.

Just to recap where we were with PCA, principal component analysis, I said that in PCA, we imagine that we have some very high dimensional data that perhaps lies approximately on some low dimensional subspace. So if you had the data set like this, you might find that that's the first principal component of the data, and that's the second component of this 2-D data.

To summarize the algorithm, we have three steps. The first step of PCA was to normalize the data to zero mean and [inaudible]. So tracked out the means of your training examples. So it now has zero means, and then normalize each of your features so that the variance of each feature is now one.

The next step was [inaudible] computical variance matrix of your zero mean data. So you compute it as follows. The sum of all the products, and then you find the top K eigen vectors of sigma. So last time we saw the applications of this. For example, one of the applications was to eigen faces where each of your training examples, XI, is an image. So if you have 100 by 100 images, if your pictures of faces are 100 pixels by 100 pixels, then each of your training examples, XI, will be a 10,000 dimensional vector, corresponding to the 10,000 grayscale intensity pixel values.

There are 10,000 pixel values in each of your 100 by 100 images. So the eigen faces application was where the training example comprised pictures of faces of people. Then we ran PCA, and then to measure the distance between say a face here and a face there, we would project both of the face images onto the subspace and then measure the distance along the subspace. So in eigen faces, you use something like 50 principle components.

So the difficulty of working with problems like these is that in step two of the algorithm, we construct the covariance matrix sigma. The covariance matrix now becomes a 10,000 by 10,000 dimensional matrix, which is huge. That has 100 million entries, which is huge.

So let's apply PCA to very, very high dimensional data, used as a point of reducing the dimension. But step two of this algorithm had this step where you were constructing [inaudible]. So this extremely large matrix, which you can't do. Come back to this in a second.

It turns out one of the other frequently-used applications of PCA is actually to text data. So here's what I mean. Remember our vectorial representation of emails? So this is way back when we were talking about supervised learning algorithms for a stand classification. You remember I said that given a piece of email or given a piece of text

document, you can represent it using a very high-dimensional vector by taking – writing down a list of all the words in your dictionary. Somewhere you had the word learn, somewhere you have the word study and so on.

Depending on whether each word appears or does not appear in your text document, you put either a one or a zero there. This is a representation we use on an electrode five or electrode six for representing text documents for when we're building [inaudible] based classifiers for [inaudible].

So it turns out one of the common applications of PCA is actually this text data representations as well. When you apply PCA to this sort of data, the resulting algorithm, it often just goes by a different name, just latent semantic indexing. For the sake of completeness, I should say that in LSI, you usually skip the preprocessing step.

For various reasons, in LSI, you usually don't normalize the mean of the data to one, and you usually don't normalize the variance of the features to one. These are relatively minor differences, it turns out, so it does something very similar to PCA.

Normalizing the variance to one for text data would actually be a bad idea because all the words are – because that would have the affect of dramatically scaling up the weight of rarely occurring words. So for example, the word aardvark hardly ever appears in any document. So to normalize the variance of the second feature to one, you end up – you're scaling up the weight of the word aardvark dramatically. I don't understand why [inaudible].

So let's see. [Inaudible] the language, something that we want to do quite often is, give it two documents, XI and XJ, to measure how similar they are. So for example, I may give you a document and ask you to find me more documents like this one. We're reading some article about some user event of today and want to find out what other news articles there are. So I give you a document and ask you to look at all the other documents you have in this large set of documents and find the documents similar to this.

So this is typical text application, so to measure the similarity between two documents in XI and XJ, [inaudible] each of these documents is represented as one of these high-dimensional vectors. One common way to do this is to view each of your documents as some sort of very high-dimensional vector. So these are vectors in the very high-dimensional space where the dimension of the vector is equal to the number of words in your dictionary.

So maybe each of these documents lives in some 50,000-dimension space, if you have 50,000 words in your dictionary. So one nature of the similarity between these two documents that's often used is what's the angle between these two documents. In particular, if the angle between these two vectors is small, then the two documents, we'll consider them to be similar. If the angle between these two vectors is large, then we consider the documents to be dissimilar.

So more formally, one commonly used heuristic, the national language of processing, is to say that the similarity between the two documents is a co-sine of the angle theta between them. For similar values, anyway, the co-sine is a decreasing function of theta. So the smaller the angle between them, the larger the similarity. The co-sine between two vectors is, of course, just [inaudible] divided by – okay? That's just the linear algebra or the standard geometry definition of the co-sine between two vectors.

Here's the intuition behind what LSI is doing. The hope, as usual, is that there may be some interesting axis of variations in the data, and there maybe some other axis that are just noise. So by projecting all of your data on lower-dimensional subspace, the hope is that by running PCA on your text data this way, you can remove some of the noise in the data and get better measures of the similarity between pairs of documents.

Let's just delve a little deeper into those examples to convey more intuition about what LSI is doing. So look further in the definition of the co-sine similarity measure. So the numerator or the similarity between the two documents was this inner product, which is therefore sum over K, XIK, XJK. So this inner product would be equal to zero if the two documents have no words in common. So this is really – sum over K – indicator of whether documents, I and J, both contain the word, K, because I guess XIK indicates whether document I contains the word K, and XJK indicates whether document J contains the word, K.

So the product would be one only if the word K appears in both documents. Therefore, the similarity between these two documents would be zero if the two documents have no words in common. For example, suppose your document, XI, has the word study and the word XJ, has the word learn. Then these two documents may be considered entirely dissimilar.

[Inaudible] effective study strategies. Sometimes you read a news article about that. So you ask, what other documents are similar to this? If there are a bunch of other documents about good methods to learn, than there are words in common. So similarity [inaudible] is zero.

So here's a cartoon of what we hope [inaudible] PCA will do, which is suppose that on the horizontal axis, I plot the word learn, and on the vertical access, I plot the word study. So the values take on either the value of zero or one. So if a document contains the words learn but not study, then it'll plot that document there. If a document contains neither the word study nor learn, then it'll plot that at zero, zero.

So here's a cartoon behind what PCA is doing, which is we identify lower dimensional subspace. That would be sum – eigen vector, we get out of PCAs. Now, supposed we have a document about learning. We have a document about studying. The document about learning points to the right. Document about studying points up. So the inner product, or the co-sine angle between these two documents would be – excuse me. The inner product between these two documents will be zero. So these two documents are entirely unrelated, which is not what we want.

Documents about study, documents about learning, they are related. But we take these two documents, and we project them onto this subspace. Then these two documents now become much closer together, and the algorithm will recognize that when you say the inner product between these two documents, you actually end up with a positive number. So LSI enables our algorithm to recognize that these two documents have some positive similarity between them.

So that's just intuition about what PCA may be doing to text data. The same thing goes to other examples and the words study and learn. So you have – you find a document about politicians and a document with the names of prominent politicians. That will also bring the documents closer together, or just any related topics, they end up [inaudible] points closer together and just lower dimensional space.

Question about this? Interviewee:

[Inaudible].

Instructor (Andrew Ng):Which ones? This one? Interviewee:

No, the line.

Instructor (Andrew Ng):Oh, this one. Oh, yes. Thank you. [Inaudible]. So let's talk about how to actually implement this now. Okay. How many of you know what an SVD or single value decomposition is? Wow, that's a lot of you. That's a lot more than I thought. Curious. Did you guys learn it as under grads or as graduate students?

All right. Let me talk about it anyway. I wasn't expecting so many of you to know what SVD is, but I want to get this on tape, just so everyone else can learn about this, too. So I'll say a little bit about how to implement PCA. The problem I was eluding to just now was that when you have these very high-dimensional vectors, than sigma is a large matrix.

In particular, for our text example, if the vectors XI are 50,000 dimensional, then the covariance matrix will be 50,000 dimensional by 50,000 dimensional, which is much too big to represent explicitly. I guess many of you already know this, but I'll just say it anyway. It turns out there's another way to implement PCA, which is if A is any N by N matrix, than one of the most remarkable results of linear algebra is that the matrix, A, can be decomposed into a singular value decomposition.

What that means is that the matrix, A, which is N by N, can always be decomposed into a product of three matrixes. U is N by N, D is a square matrix, which is N by N, and V is also N by N. D is going to be diagonal. Zeros are on the off-diagonals, and the values sigma I are called the singular values of the matrix A.

Almost all of you said you learned this as a graduate student, rather than as an under grad, so it turns out that when you take a class in undergraduate linear algebra, usually

you learn a bunch of decomposition. So you usually learn about the QLD composition, maybe the LU factorization of the matrixes. Most under grad courses don't get to talk about singular value decompositions, but at least in – almost everything I do in machine learning, you actually find that you end up using SVDs much more than any of the decompositions you learned in typical under grad linear algebra class.

So personally, I [inaudible] an SVD dozens of times in the last year, but LU and QRD compositions, I think I used the QRD composition once and an LU decomposition in the last year. So let's see. I'll say a bit more about this. So I'm going to draw the picture, I guess.

For example, if A is an N by N matrix, it can be decomposed into another matrix, U, which is also N by N. It's the same size, D, which is N by N. Another square matrix, V transpose, which is also N by N. Furthermore, in a singular value decomposition, the columns of the matrix, U, will be the eigen vectors of A transpose, and the columns of V will be the eigen vectors of A transpose A.

To compute it, you just use the SVD commands in Matlab or Octave. Today, say the art in numerical linear algebra is that SVD, singular value decompositions, and matrixes can be computed extremely [inaudible]. We've used a package like Matlab or Octave to compute, say, the eigen vectors of a matrix. So if SVD routines are even more numerically stable than eigen vector routines for finding eigen vector in the matrix. So you can safely use a routine like this, and similar to the way they use a square root command without thinking about how it's computed.

You can compute the square root of something and just not worry about it. You know the computer will give you the right answer. For most reasonably-sized matrixes, even up to thousands by thousands matrixes, the SVD routine, I think of it as a square root function. If you call it, it'll give you back the right answer. You don't have to worry too much about it.

If you have extremely large matrixes, like a million by a million matrixes, I might start to worry a bit, but a few thousand by a few thousand matrixes, this is implemented very well today. Interviewee:

[Inaudible].

Instructor (Andrew Ng):What's the complexity of SVD? That's a good question. I actually don't know. I want to guess it's roughly on the order of N-cubed. I'm not sure. [Inaudible] algorithms, so I think – I don't know what's known about the conversion of these algorithms.

The example I drew out was for a facts matrix, and a matrix is [inaudible]. In the same way, you can also call SVD on the tall matrix, so it's taller than it's wide. It would decompose it into – okay? A product of three matrixes like that.

The nice thing about this is that we can use it to compute eigen vectors and PCA very efficiently. In particular, a covariance matrix sigma was this. It was the sum of all the products, so if you go back and recall the definition of the design matrix – I think I described this in lecture two when we derived the close form solution to these squares [inaudible] these squares. The design matrix was this matrix where I took my examples and stacked them in rows. They call this the design matrix [inaudible].

So if you construct the design matrix, then the covariance matrix sigma can be written just X transposing. That's X transposed, and [inaudible]. Okay? I hope you see why the X transpose X gives you the sum of products of vectors. If you aren't seeing this right now, just go home and convince yourself [inaudible] if it's true.

To get the top K eigen vectors of sigma, you would take sigma and decompose it using the – excuse me. You would take the matrix X, and you would compute as SVD. So you get USV transpose. Then the top three columns of U are the top K eigen vectors of X transpose X, which is therefore, the top K eigen vectors of your covariance matrix sigma.

So in our examples, the design matrix may be, say R. If you have 50,000 words in your dictionary, than the design matrix would be RM by 50,000. [Inaudible] say 100 by 50,000, if you have 100 examples. So X would be quite tractable to represent and compute the SVD, whereas the matrix sigma would be much harder to represent. This is 50,000 by 50,000. So this gives you an efficient way to implement PCA.

The reason I want to talk about this is in previous years, I didn't talk [inaudible]. The class projects, I found a number of students trying to implement SVD on huge problems and [inaudible], so this is a much better to implement PCA if you have extremely high dimensional data. If you have low dimensional data, if you have 50 or 100 dimensional data, then computing sigma's no problem. You can do it the old way, but otherwise, use the SVD to implement this.

Questions about this?

The last thing I want to say is that in practice, when you want to implement this, I want to say a note of caution. It turns out that for many applications of – let's see. When you apply SVD to these wide – yeah. Interviewee:

Just a quick question. Are the top K columns of U or V because X transposed X is V transpose, right?

Instructor (Andrew Ng):Let's see. Oh, yes. I think you're right. I think you're right. Let's see. Is it top K columns of U or top K of V? Yeah, I think you're right. Is that right? Something bothers me about that, but I think you're right. Interviewee:

[Inaudible], so then X transpose X should be VDD. X is UDV, so X transpose X would be –

Instructor (Andrew Ng):[Inaudible]. If anyone thinks about this and has another opinion, let me know, but I think you're right. I'll make sure I get the details and let you know. Everyone's still looking at that. Tom, can you figure out the right answer and let me know? Male Speaker:

That sounds right.

Instructor (Andrew Ng):Okay. Cool. Okay. So just one last note, a note of caution. It turns out that in this example, I was implementing SVD with a wide matrix. So the matrix X was N by N. It turns out when you find the SVD decomposition of this, it turns out that – let's see. Yeah, I think you're definitely right. So it turns out that we find the SVD of this, the right-most portion of this block of this matrix would be all zeros.

Also, when you compute the matrix, D, a large part of this matrix would be zeros. You have the matrix D transpose. So depending on what convention you use, for example, I think Matlab actually uses a convention of just cutting off the zero elements. So the Matlab uses the convention of chopping off the right-most half of the U matrix and chopping off the bottom portion of the D matrix.

I'm not sure if this even depends on the version of Matlab, but when you call SVD on Matlab or some other numerical algebra packages, there's slightly different conventions of how to define your SVD when the matrix is wider than it is tall. So just watch out for this and make sure you map whatever convention your numerical algebra library uses to the original computations.

It turns out if you turn Matlab [inaudible] or you're writing C code. There are many scientific libraries that can compute SVDs for you, but they're just slightly different in conventions for the dimensions of these matrixes. So just make sure you figure this out for the package that you use.

Finally, I just want to take the unsupervised learning algorithms we talked about and just put a little bit of broader context. This is partly in response to the questions I've gotten from students in office hours and elsewhere about when to use each of these algorithms. So I'm going to draw a two by two matrix. This is a little cartoon that I find useful.

One of the algorithms we talked about earlier, right before this, was factor analysis, which was – it was – I hope you remember that picture I drew where I would have a bunch of point Z on the line. Then I had these ellipses that I drew. I hope you remember that picture. This was a factor analysis model which models the density effects [inaudible], right?

It was also a PCA, just now. So the difference between factor analysis and PCA, the way I think about it, is that factor analysis is a density estimation algorithm. It tries to model the density of the training example's X. Whereas PCA is not a probabilistic algorithm. In particular, it does not endow your training examples of any probabilistic distributions and directly goes to find the subspace.

So in terms of when to use factor analysis and when to use PCA, if your goal is to reduce the dimension of the data, if your goal is to find the subspace that the data lies on, then PCA directly tries to find the subspace. I think I would tend to use PCA.

Factor analysis, it sort of assumes the data lies on a subspace. Let me write a subspace here. So both of these algorithms sort of assume the data maybe lies close or on some low dimensional subspace, but fundamentally, factor analysis, I think of it as a density estimation algorithm. So that has some very high dimensional distribution. I want to model P of X, then the factor analysis is the algorithm I'm more inclined to use.

So even though you could in theory, I would tend to avoid trying to use factor analysis to identify a subspace the data set lies on. So [inaudible], if you want to do anomaly detection, if you want to model P of X so that if you have a very low probability of N, you can factor an anomaly, then I would tend to use factor analysis to do that density estimation.

So factor analysis and PCA are both algorithms that assume that your data lies in the subspace. The other cause of algorithms we talked about was algorithms that assumes the data lies in clumps or that the data has a few coherence to groups. So let me just fill in the rest of this picture. So if you think your data lies in clumps or lies in groups, and if it goes [inaudible] density estimation, then I would tend to use a mixture of [inaudible] algorithm.

But again, you don't necessarily want to endow your data of any probably semantics, so if you just want to find the clumps of the groups, then I'd be inclined to use a [inaudible] algorithm. So haven't seen anyone else draw this picture before, but I tend to organize these things this way in my brain. Hopefully this helps guide when you might use each of these algorithms as well, depending on whether you believe the data might lie in the subspace or whether it might bind in clumps or groups.

All right. That wraps up the discussion on PCA. What I want to do next is talk about independent component analysis, or ICA. Yeah. Interviewee:

I have a question about the upper right [inaudible]. So once you have all of the eigen vectors, [inaudible] how similar is feature I to feature J. You pick some eigen vector, and you take some dot products between the feature I and feature J and the eigen vector. But there's a lot of eigen vectors to choose from.

Instructor (Andrew Ng):Right. So Justin's question was having found my eigen vectors, how do I choose what eigen vector to use to measure distance. I'm going to start this up. So the answer is really – in this cartoon, I would avoid thinking about eigen vectors one other time. A better way to view this cartoon is that this is actually – if I decide to choose 100 eigen vectors, this is really 100 D subspace.

So I'm not actually projecting my data onto one eigen vector. This arrow, this cartoon, this denotes the 100-dimensional subspace [inaudible] by all my eigen vectors. So what I

actually do is project my data onto the span, the linear span of eigen vectors. Then I measure distance or take inner products of the distance between the projections of the two points of the eigen vectors. Okay.

So let's talk about ICA, independent component analysis. So whereas PCA was an algorithm for finding what I call the main axis of variations of data, in ICA, we're going to try find the independent of components of variations in the data. So switch it to the laptop there, please.

We'll just take a second to motivate that. I'm going to do so by – although if you put on the – okay. This is actually a slide that I showed in lecture one of the cocktail party problem. Suppose you have two speakers at a cocktail party, and you have two microphones in the room, overlapping sets of two conversations. Then can you separate out the two original speaker sources? So I actually played this audio as well in the very first lecture, which is suppose microphone one records this. [Recording]

Instructor (Andrew Ng):So the question is, these are really two speakers, speaking independently of each other. So each speaker is outputting a series of sound signals as independent of the other conversation going on in the room. So this being an supervised learning problem, the question is, can we take these two microphone recordings and feed it to an algorithm to find the independent components in this data? This is the output when we do so. [Recording]

Instructor (Andrew Ng):This is the other one. [Recording]

Instructor (Andrew Ng):Just for fun. [Inaudible]. These are audio clips I got from [inaudible]. Just for fun, let me play the other ones as well. This is overlapping microphone one. [Recording]

Instructor (Andrew Ng):Here's microphone two. [Recording]

Instructor (Andrew Ng):So given this as input, here's output one. [Recording]

Instructor (Andrew Ng):It's not perfect, but it's largely cleaned up the music. Here's number two. [Recording]

Instructor (Andrew Ng):Okay. Switch back to [inaudible], please. So what I want to do now is describe an algorithm that does that. Before I actually jump into the algorithm, I want to say two minutes of CDF, so cumulative distribution functions. I know most of you know what these are, but I'm just going to remind you of what they are.

Let's say you have a one-D random variable S. So suppose you have a random variable, S, and suppose it has a property density function [inaudible]. Then the CDF is defined as a function, or rather as F, which is the probability that the random variable, S, is less than the value given by that lower-case value, S. For example, if this is your [inaudible]

density, than the density of the [inaudible] usually to note it lower-case phi. That's roughly a bell-shaped density.

Then the CDF or the Gaussian will look something like this. There'll be a capital function pi. So if I pick a value S like that, then the height of this – this is [inaudible] probability that my Gaussian random variable is less than that value there. In other words, the height of the function at that point is less than the area of the Gaussian density, up to the point S. As you move further and further to the right, this function will approach one, as you integrate more and more of this area of the Gaussian.

So another way to write F of S is the integral, the minus infinity to S of the density, DT. So something that'll come later is suppose I have a random variable, S, and I want to model the distribution of the random variable, S. So one thing I could do is I can specify what I think the density is. Or I can specify what the CDF is. These are related by this equation. F is the integral of P of S.

You can also recover the density by taking the CDF and taking the derivative. So F prime, take the derivative of the CDF, you get back the density. So this has come up in the middle of when I derive ICA, which is that there'll be a step where they need to assume a distribution for random variable, S. I can either specify the density for S directly, or I can specify the CDF. I choose to specify the CDF.

It has to be some function increasing from zero to one. So you can choose any function that looks like that, and in particular, pulling functions out of a hat that look like that. You can, for instance, choose a sigmoid function of CDF. That would be one way of specifying the distribution of the densities for the random variable S. So this will come up later.

Just [inaudible], just raise your hand if that is familiar to you, if you've seen that before. Great.

So let's start to derive our RCA, or our independent component analysis algorithm. Let's assume that the data comes from N original sources. So let's say there are N speakers in a cocktail party. So the original sources, I'm going to write as a vector, S as in RN. So just to be concrete about what I mean about that, I'm going to use SIJ to denote the signal from speaker J at time I.

Here's what I mean. So what is sound? When you hear sound waves, sound is created by a pattern of expansions and compressions in air. So the way you're hearing my voice is my mouth is causing certain changes in the air pressure, and then your ear is hearing my voice as detecting those changes in air pressure. So what a microphone records, what my mouth is generating, is a pattern. I'm going to draw a cartoon, I guess. Changes in air pressure. So this is what sound is.

You look at a microphone recording, you see these roughly periodic signals that comprise of changes in air pressure over time as the air pressure goes above and below some

baseline air pressure. So this is what the speech signal looks like, say. So this is speaker one. Then what I'm saying is that – this is some time, T.

What I'm saying is that the value of that point, I'm going to denote as S, super script T, sub script one. Similarly, speaker two, it's outputting some sound wave. Speaker voice will play that. It'll actually sound like a single tone, I guess. So in the same way, at the same time, T, the value of the air pressure generated by speaker two, I'll denote as ST 2.

So we observe XI equals A times SI, where these XIs are vectors in RN. So I'm going to assume that I have N microphones, and each of my microphones records some linear combination of what the speakers are saying. So each microphone records some overlapping combination of what the speakers are saying.

For example, XIJ, which is – this is what microphone J records at time, I. So by definition of the matrix multiplication, this is sum of AIKSJ. Oh, excuse me. Okay? So what my J – sorry. So what my J microphone is recording is some linear combination of all of the speakers. So at time I, what microphone J is recording is some linear combination of what all the speakers are saying at time I. So K here indexes over the N speakers.

So our goal is to find the matrix, W, equals A inverse, and just defining W that way. So we can recover the original sources as a linear combination of our microphone recordings, XI. Just as a point of notation, I'm going to write the matrix W this way. I'm going to use lower case W subscript one, subscript two and so on to denote the roles of this matrix, W.

Let's see. So let's look at why IC is possible. Given these overlapping voices, let's think briefly why it might be possible to recover the original sources. So for the next example, I want to say – let's say that each of my speakers outputs – this will sound like white noise. Can I switch the laptop display, please? For this example, let's say that each of my speakers outputs uniform white noise. So if that's the case, these are my axis, S1 and S2. This is what my two speakers would be uttering.

The parts of what they're uttering will look like a line in a square box if the two speakers are independently outputting uniform minus one random variables. So this is part of S1 and S2, my original sources. This would be a typical sample of what my microphones record. Here, at the axis, are X1 and X2. So these are images I got from [inaudible] on ICA. Given a picture like this, you can sort of look at this box, and you can sort of tell what the axis of this parallelogram are. You can figure out what linear transformation would transform the parallelogram back to a box.

So it turns out there are some inherent ambiguities in ICA. I'll just say what they are. One is that you can't recover the original indexing of the sources. In particular, if I generated the data for speaker one and speaker two, you can run ICA, and then you may end up with the order of the speakers reversed. What that corresponds to is if you take this

picture and you flip this picture along a 45-degree axis. You take a 45-degree axis and reflect this picture across the 45-degree axis, you'll still get a box.

So there's no way for the algorithms to tell which was speaker No. 1 and which was speaker No. 2. The numbering or the ordering of the speakers is ambiguous. The other source of ambiguity, and these are the only ambiguities in this example, is the sign of the sources. So given my speakers' recordings, you can't tell whether you got a positive SI or whether you got back a negative SI.

In this picture, what that corresponds to is if you take this picture, and you reflect it along the vertical axis, if you reflect it along the horizontal axis, you still get a box. You still get back [inaudible] speakers. So it turns out that in this example, you can't guarantee that you've recovered positive SI rather than negative SI.

So it turns out that these are the only two ambiguities in this example. What is the permutation of the speakers, and the other is the sign of the speakers. Permutation of the speakers, there's not much you can do about that. It turns out that if you take the audio source and if you flip the sign, and you take negative S, and if you play that through a microphone it'll sound indistinguishable. So for many of the applications we care about, the sign as well as the permutation is ambiguous, but you don't really care about it.

Let's switch back to chalk board, please. It turns out, and I don't want to spend too much time on this, but I do want to say it briefly. It turns out the reason why those are the only sources of ambiguity – so the ambiguities were the permutation of the speakers and the signs. It turns out that the reason these were the only ambiguities was because the SIJs were non-Gaussian. I don't want to spend too much time on this, but I'll say it briefly.

Suppose my original sources, S1 and S2, were Gaussian. So suppose SI is Gaussian, would mean zero and identity covariance. That just means that each of my speakers outputs a Gaussian random variable. Here's a typical example of Gaussian data.

You will recall the contours of a Gaussian distribution with identity covariants looks like this, right? The Gaussian is a spherically symmetric distribution. So if my speakers were outputting Gaussian random variables, than if I observe a linear combination of this, there's actually no way to recover the original distribution because there's no way for me to tell if the axis are at this angle or if they're at that angle and so on. The Gaussian is a rotationally symmetric distribution, so I would no be able to recover the orientation in the rotation of this.

So I don't want to prove this too much. I don't want to spend too much time dwelling on this, but it turns out if your source is a Gaussian, then it's actually impossible to do ICA. ICA relies critically on your data being non-Gaussian because if the data were Gaussian, then the rotation of the data would be ambiguous. So regardless of how much data you have, even if you had infinitely large amounts of data, you would not be able to recover the matrix A or W.

Let's go ahead and divide the algorithm. To do this, I need just one more result, and then the derivation will be three lines. [Inaudible] many variables as N, which is the joint vector of the sound that all of my speakers that are emitting at any time. So let's say the density of S is P subscript S, capital S. So my microphone recording records S equals AS, equals W inverse S. Equivalently, S equals W sign of X.

So let's think about what is the density of X. So I have P of S. I know the density of S, and X is a linear combination of the S's. So let's figure out what is the density of X. One thing we could do is figure out what S is. So this is just – apply the density of S to W of S. So let's see. This is the probability of S, so we just figure out what S is.

S is W times X, so the probability of S is W times X, so the probability of X must be [inaudible]. So this is wrong. It turns out you can do this for probably mass functions but not for continuous density. So in particular, it's not correct to say that the probability of X is – well, you just figure out what S is.

Then you say the probability of S is applied to that. This is wrong. You can't do this with densities. You can't say the probability of S is that because it's a property density function. In particular, the right formula is the density of S applied to W times X, times the determinant of the matrix, W. Let me just illustrate that with an example.

Let's say the density for S is that. In this example, S is uniform over the unit interval. So the density for S looks like that. It's just density for the uniform distribution of zero one. So let me let X be equal to two times S. So this means A equals two. W equals one half. So if S is a uniform distribution over zero, one, then X, which is two times that, will be the uniform distribution over the range from zero to two.

So the density for X will be – that's one, that's two, that's one half, and that's one. Okay? Density for X will be indicator zero [inaudible] for X [inaudible] two times W, times one half.

So does that make sense? [Inaudible] computer density for X because X is now spread out across a wider range. The density of X is now smaller, and therefore, the density of X has this one half term here. Okay?

This is an illustration for the case of one-dimensional random variables, or S and X of one D. I'm not going to show it, but the generalization of this to vector value random variables is that the density of X is given by this times the determinant of the matrix, W. Over here, I showed the one dimensional [inaudible] generalization.

So we're nearly there. Here's how I can implement ICA. So my distribution on S, so I'm going to assume that my density on S is given by this as a product over the N speakers of the density – the product of speaker I emitting a certain sound. This is a product of densities. This is a product of distributions because I'm going to assume that the speakers are having independent conversations.

So the SI's independent for different values of I. So by the formula we just worked out, the density for X would be equal to that. I'll just remind you, W was A inverse. It was this matrix I defined previously so that SI equals WI [inaudible] X. So that's what's in there.

To complete my formulation for this model, the final thing I need to do is choose a density for what I think each speaker is saying. I need to assume some density over the sounds emitted by an individual speaker. So following the discussion I had right when the [inaudible] ICA, one thing I could do is I could choose the density for S, or equivalently, I could choose the CDF, the cumulative distribution function for S.

In this case, I'm going to choose a CDF, probably for historical reasons and probably for convenience. I need to choose the CDF for S, so what that means is I just need to choose some function that increases from zero to what. I know I can't choose a Gaussian because we know you can't do ICA on Gaussian data. So I need some function increasing from zero to one that is not the cumulative distribution function for a Gaussian distribution.

So what other functions do I know that increase from zero to one? I just choose the CDF to be the sigmoid function. This is a commonly-made choice that is made for convenience. There is actually no great reason for why you choose a sigmoid function. It's just a convenient function that we all know and are familiar with that happens to increase from zero to one.

When you take the derivative of the sigmoid, and that will give you back your density. This is just not Gaussian. This is the main virtue of choosing the sigmoid. So there's really no rational for the choice of sigma. Lots of other things will work fine, too. It's just a common, reasonable default.

It turns out that one reason the sigma works well for a lot of data sources is that if this is the Gaussian. If you actually take the sigmoid and you take its derivative, you find that the sigmoid has [inaudible] than the Gaussian. By this I mean the density of the sigmoid dies down to zero much more slowly than the Gaussian. The magnitudes of the tails dies down as E to the minus S squared. For the sigmoid, the tails look like E to the minus S. So the tails die down as E to the minus S, around E to the minus S squared.

It turns out that most distributions of this property with [inaudible] tails, where the distribution decays to zero relatively slowly compared to Gaussian will work fine for your data. Actually, one other choice you can sometimes us is what's called the Laplacian distribution, which is that. This will work fine, too, for many data sources.

Sticking with the sigmoid for now, I'll just write down the algorithm in two steps. So given my training set, and as you show, this is an unlabeled training set, I can write down the log likelihood of my parameters. So that's – assembled my training examples, log of – times that. So that's my log likelihood. To learn the parameters, W, of this model, I can use the [inaudible] assent, which is just that. It turns out, if you work through the math, let's see. If P of S is equal to the derivative of the sigmoid, then if you just work through

the math to compute the [inaudible] there. You've all done this a lot of times. I won't bother to show the details. You find that is equal to this.

Okay? That's just – you can work those out yourself. It's just math to compute the derivative of this with respect to W. So to summarize, given the training set, here's my [inaudible] update rule. So you run the [inaudible] to learn the parameters W. After you're done, you then output SI equals WXI, and you've separated your sources of your data back out into the original independent sources.

Hopefully up to only a permutation and a plus/minus sign ambiguity. Okay? So just switch back to the laptop, please? So we'll just wrap up with a couple of examples of applications of ICA.

This is actually a picture of our TA, Katie. So one of the applications of ICA is to process various types of [inaudible] recording data, so [inaudible]. This is a picture of a EEG cap, in which there are a number of electrodes you place on the – in this case, on Katie's brain, on Katie's scalp. So where each electrode measures changes in voltage over time on the scalp. On the right, it's a typical example of [inaudible] data where each electrode measures – just changes in voltage over time.

So the horizontal axis is time, and the vertical axis is voltage. So here's the same thing, blown up a little bit. You notice there are artifacts in this data. Where the circle is, where the data is circled, all the electrodes seem to measure in these very synchronized recordings. It turns out that we look at [inaudible] data as well as a number of other types of data, there are artifacts from heartbeats and from human eye blinks and so on.

So the cartoonist, if you imagine, placing the electrodes, or microphones, on my scalp, then each microphone is recording some overlapping combination of all the things happening in my brain or in my body. My brain has a number of different processes going on. My body's [inaudible] going on, and each electrode measures a sum of the different voices in my brain. That didn't quite come out the way I wanted it to.

So we can just take this data and run ICA on it and find out one of the independent components, what the independent process are going on in my brain. This is an example of running ICA. So you find that a small number of components, like those shown up there, they correspond to heartbeat, where the arrows – so those are very periodic signals. They come on occasionally and correspond to [inaudible] components of heartbeat.

You also find things like an eye blink component, corresponding to a sigmoid generated when you blink your eyes. By doing this, you can then subtract out the heartbeat and the eye blink artifacts from the data, and now you get much cleaner ICA data – get much cleaner EEG readings. You can do further scientific studies.

So this is a pretty commonly used preprocessing step that is a common application of ICA. [Inaudible] example is the application, again, from [inaudible]. As a result of running ICA on natural small image patches. Suppose I take natural images and run ICA

on the data and ask what are the independent components of data. It turns out that these are the bases you get. So this is a plot of the sources you get.

This algorithm is saying that a natural image patch shown on the left is often expressed as a sum, or a linear combination, of independent sources of things that make up images. So this model's natural images is generated by independent objects that generate different ages in the image.

One of the fascinating things about this is that, similar to neuroscience, this has also been hypothesized as a method for how the human brain processes image data. It turns out, this is similar, in many ways, to computations happening in early visual processing in the human brain, in the mammalian brain. It's just interesting to see ages are the independent components of images.

Are there quick questions, because I'm running late. Quick questions before I close? Interviewee:

[Inaudible] square matrix?

Instructor (Andrew Ng):Oh, yes. For the algorithms I describe, I assume A is a square matrix. It turns out if you have more microphones than speakers, you can also apply very similar algorithms. If you have fewer microphones than speakers, there's sort of an open research problem. The odds are that if you have one male and one female speaker, but one microphone, you can sometimes sort of separate them because one is high, one is low.

If you have two male speakers or two female speakers, then it's beyond the state of the art now to separate them with one microphone. It's a great research program.

Okay. Sorry about running late again. Let's close now, and we'll continue reinforcement learning next [inaudible].

[End of Audio]

Duration: 80 minutes