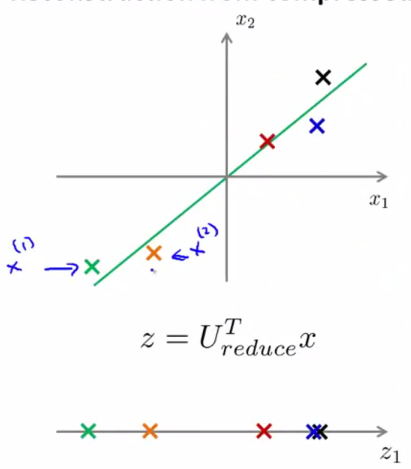
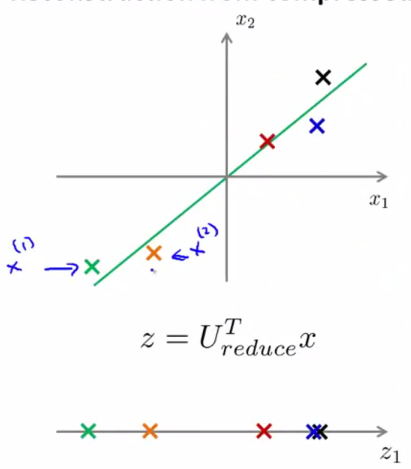
***Dimensionality Reduction***

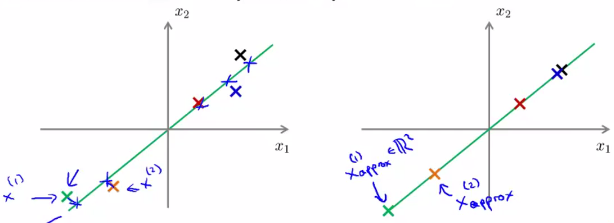
***Applying PCA***

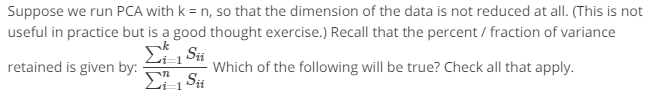
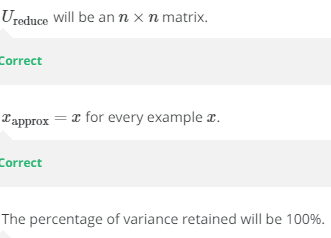
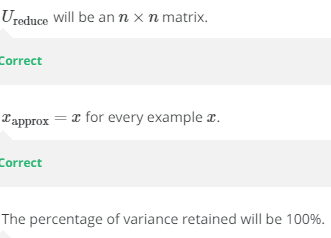
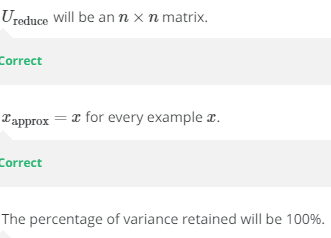
**I. RECONSTRUCTION FROM COMPRESSED REPRESENTATION**

* Since PCA is a compression algorithm, there should be a way to go back from the compressed representation back to an approximation of the original high-dimensional data.
* So given a 100-dimensional z(i), how do you go back to your 1000-dimensional original representation, x(i)
* In the PCA algorithm for 2D data, we have examples + project them onto a 1D surface + then now use a real number, say z1, to specify the location of these example points after they've been projected onto this surface.



* So, given the point z1, which is in Ɍ, how can we go back to this original 2D space/map this z1 back to some approximate representation x in Ɍ2 of whatever the original value of the data was?
* Whereas z = U(reduced)(t)\*x, if you want to go in the *opposite* direction, we write:
* **x(approx) = U(reduced)\*z**
* Remember U(reduced) is an n\*k-dimensional vector + z is a k\*1-dimensional vector, so when you multiply these, x(approx) is going to be an n\*1 or n-dimensional vector.
* The intent of PCA is if the squared projection error is not too big, this x(approx) will be close to the original value of x used to derive z in the first place.



* This is a pretty decent approximation of the original data
* This process = **reconstruction** of the original data (think of trying to reconstruct the original value of x from the compressed representation)
* 
* 

**II. CHOOSING THE NUMBER OF PRINCIPAL COMPONENTS**

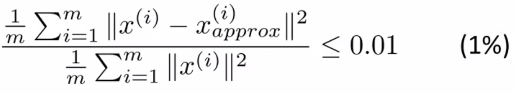
* So, in the PCA algorithm we take n dimensional features + reduce them to k dimensional feature representation, + this number k is a parameter of the PCA algorithm.
* k = the **number of principle components** **retained**
* We know PCA tries to minimize the average squared projection error (difference between original data X + projected version, X(approx)(i)) on the lower-dimensional surface.



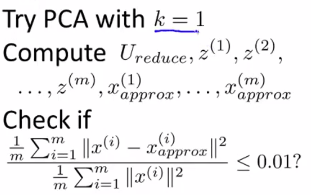
* 1:01
* **Total variation** in the data is the squared average length of examples X(i)



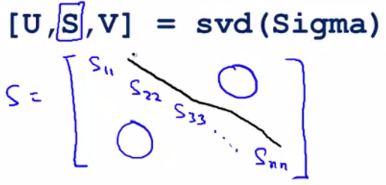
* "On average, how far are my training examples from the origin/the vector of all 0’s"
* A pretty common rule of thumb for choosing k is to choose the smallest value such that the ratio between average squared projection error and total variation is less than 0.01.



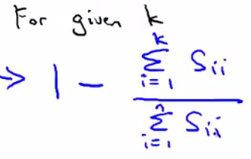
* The way most people think about choosing k is, rather than choosing k directly, being concerned with that this ratio is
* If it is 0.01, in the language of PCA we say “99% of the variance is retained“
* If using PCA + want to tell someone how many principle components you've retained, it’s more common to say “I chose k so that 99% of the variance was retained”
* "I had to 100 principle components" or "k was equal to 100 in a 1000-dimensional data" is harder to interpret
* Other common values is 0.05, or 5%, + you’d say “95% of the variance is retained”
* As low as 85% of variance contained would be a fairly typical range in values, while 95-99 is the most common range of values people use.
* For many data sets, you can often reduce the dimensions of the data significantly + still retain most/99% of the variance.
* B/c for most real-life data sets, many features are highly correlated, + it turns out to be possible to compress the data a lot + still retain 99%/95% of the variance.
* So how do you implement this? Here's 1 algorithm you might use



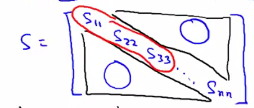
* Start off w/ k = 1, run through PCA (compute U(reduced), z1, z2, up to z(m), x(approx)1 up to x(approx)(m), then check if 99% of the variance is retained.
* If it isn't, try k = 2 and repeat
* This procedure seems horribly inefficient
* Fortunately when you implement PCA it actually, the inner computation step actually gives us a quantity that makes it much easier to compute the average squared projection error + total variance
* When you're calling **SVD** on the covariance matrix δ to get the matrices U, S, V, it also gives back a matrix S, an diagonal n\*n square matrix



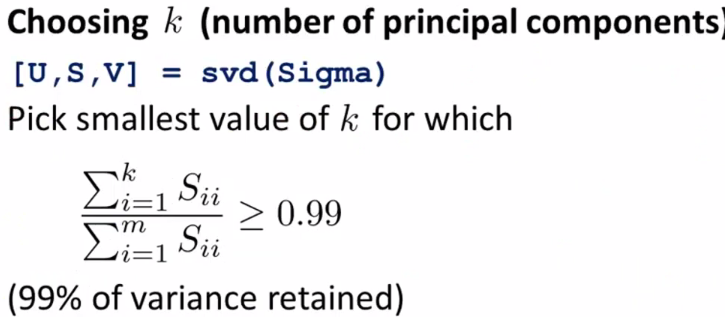
* S(1,1) to S(n,n) are the only non-zero elements of this matrix
* It turns out that for a given value of k, our PCA ratio can be computed much more simply as:

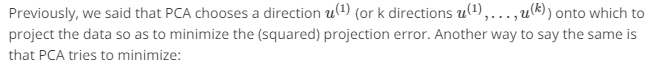


* If k = 3, we compute the sum of S(i,i) for i = 1 through i = 3



* For the denominator, we sum all the diagonal entries.
* Then we test if 1 – this ratio is less than or equal to 0.01 to be sure 99% of the variance is retained.
* Can then just slowly increase k + test this quantity to find the smallest value of k that ensures 99% of the variance is retained.
* If you do this, you only need to call SVD()once b/c it gives the S matrix
* Once you have the S matrix, you can then just keep doing this calculation by increasing k
* This procedure is much more efficient + allows you to select k w/out needing to run PCA from scratch over + over.
* Run SVD once to get all those diagonal numbers + vary k in the ratio to find the smallest value of k so that 99% of the variance is retained.



* Even if you picked k manually, if you want to explain to others what you did, a good way to explain the performance of your implementation of PCA is to compute this quantity to find the percentage of variance retained.
* If you report that number, people familiar w/ PCA get a good understanding of how well your k-dimensional representation is approximating the original data set
* 99% of variance retained = a measure of **squared reconstruction error**
* A ratio <= 0.01 gives people a good intuitive sense of whether your implementation of PCA is finding a good approximation of the original data set.
* If you apply PCA to very high-dimensional data sets, very often, just b/c data sets tend to have highly correlated features, PCA will be able to retain 99% of the variance, even while compressing the data by a very large factor.
* 
* 

**III. ADVICE FOR APPLYING PCA**

* In an earlier video, I had said that PCA can be sometimes used to speed up the running time of a learning algorithm.
* 0:07
* In this video, I'd like to explain how to actually do that, and also say some, just try to give some advice about how to apply PCA.
* 0:17
* Here's how you can use PCA to speed up a learning algorithm, and this supervised learning algorithm speed up is actually the most common use that I personally make of PCA. Let's say you have a supervised learning problem, note this is a supervised learning problem with inputs X and labels Y, and let's say that your examples xi are very high dimensional. So, lets say that your examples, xi are 10,000 dimensional feature vectors.
* 0:45
* One example of that, would be, if you were doing some computer vision problem, where you have a 100x100 images, and so if you have 100x100, that's 10000 pixels, and so if xi are, you know, feature vectors that contain your 10000 pixel intensity values, then you have 10000 dimensional feature vectors.
* 1:06
* So with very high-dimensional feature vectors like this, running a learning algorithm can be slow, right? Just, if you feed 10,000 dimensional feature vectors into logistic regression, or a new network, or support vector machine or what have you, just because that's a lot of data, that's 10,000 numbers,
* 1:24
* it can make your learning algorithm run more slowly.
* 1:27
* Fortunately with PCA we'll be able to reduce the dimension of this data and so make our algorithms run more efficiently. Here's how you do that. We are going first check our labeled training set and extract just the inputs, we're just going to extract the X's and temporarily put aside the Y's. So this will now give us an unlabelled training set x1 through xm which are maybe there's a ten thousand dimensional data, ten thousand dimensional examples we have. So just extract the input vectors
* 1:58
* x1 through xm.
* 2:00
* Then we're going to apply PCA and this will give me a reduced dimension representation of the data, so instead of 10,000 dimensional feature vectors I now have maybe one thousand dimensional feature vectors. So that's like a 10x savings.
* 2:15
* So this gives me, if you will, a new training set. So whereas previously I might have had an example x1, y1, my first training input, is now represented by z1. And so we'll have a new sort of training example,
* 2:28
* which is Z1 paired with y1.
* 2:30
* And similarly Z2, Y2, and so on, up to ZM, YM. Because my training examples are now represented with this much lower dimensional representation Z1, Z2, up to ZM. Finally, I can take this
* 2:43
* reduced dimension training set and feed it to a learning algorithm maybe a neural network, maybe logistic regression, and I can learn the hypothesis H, that takes this input, these low-dimensional representations Z and tries to make predictions.
* 2:57
* So if I were using logistic regression for example, I would train a hypothesis that outputs, you know, one over one plus E to the negative-theta transpose
* 3:07
* Z, that takes this input to one these z vectors, and tries to make a prediction.
* 3:15
* And finally, if you have a new example, maybe a new test example X. What you do is you would take your test example x,
* 3:24
* map it through the same mapping that was found by PCA to get you your corresponding z. And that z then gets fed to this hypothesis, and this hypothesis then makes a prediction on your input x.
* 3:38
* One final note, what PCA does is it defines a mapping from x to z and this mapping from x to z should be defined by running PCA only on the training sets. And in particular, this mapping that PCA is learning, right, this mapping, what that does is it computes the set of parameters. That's the feature scaling and mean normalization. And there's also computing this matrix U reduced. But all of these things that U reduce, that's like a parameter that is learned by PCA and we should be fitting our parameters only to our training sets and not to our cross validation or test sets and so these things the U reduced so on, that should be obtained by running PCA only on your training set. And then having found U reduced, or having found the parameters for feature scaling where the mean normalization and scaling the scale that you divide the features by to get them on to comparable scales. Having found all those parameters on the training set, you can then apply the same mapping to other examples that may be In your cross-validation sets or in your test sets, OK? Just to summarize, when you're running PCA, run your PCA only on the training set portion of the data not the cross-validation set or the test set portion of your data. And that defines the mapping from x to z and you can then apply that mapping to your cross-validation set and your test set and by the way in this example I talked about reducing the data from ten thousand dimensional to one thousand dimensional, this is actually not that unrealistic. For many problems we actually reduce the dimensional data. You
* 5:17
* know by 5x maybe by 10x and still retain most of the variance and we can do this barely effecting the performance,
* 5:23
* in terms of classification accuracy, let's say, barely affecting the classification accuracy of the learning algorithm. And by working with lower dimensional data our learning algorithm can often run much much faster. To summarize, we've so far talked about the following applications of PCA.
* 5:41
* First is the compression application where we might do so to reduce the memory or the disk space needed to store data and we just talked about how to use this to speed up a learning algorithm. In these applications, in order to choose K, often we'll do so according to, figuring out what is the percentage of variance retained, and so for this learning algorithm, speed up application often will retain 99% of the variance. That would be a very typical choice for how to choose k. So that's how you choose k for these compression applications.
* 6:17
* Whereas for visualization applications
* 6:20
* while usually we know how to plot only two dimensional data or three dimensional data,
* 6:26
* and so for visualization applications, we'll usually choose k equals 2 or k equals 3, because we can plot only 2D and 3D data sets.
* 6:34
* So that summarizes the main applications of PCA, as well as how to choose the value of k for these different applications.
* 6:42
* I should mention that there is often one frequent misuse of PCA and you sometimes hear about others doing this hopefully not too often. I just want to mention this so that you know not to do it. And there is one bad use of PCA, which iss to try to use it to prevent over-fitting.
* 7:00
* Here's the reasoning.
* 7:01
* This is not a great way to use PCA, but here's the reasoning behind this method, which is,you know if we have Xi, then maybe we'll have n features, but if we compress the data, and use Zi instead and that reduces the number of features to k, which could be much lower dimensional. And so if we have a much smaller number of features, if k is 1,000 and n is 10,000, then if we have only 1,000 dimensional data, maybe we're less likely to over-fit than if we were using 10,000-dimensional
* 7:33
* data with like a thousand features. So some people think of PCA as a way to prevent over-fitting. But just to emphasize this is a bad application of PCA and I do not recommend doing this. And it's not that this method works badly. If you want to use this method to reduce the dimensional data, to try to prevent over-fitting, it might actually work OK. But this just is not a good way to address over-fitting and instead, if you're worried about over-fitting, there is a much better way to address it, to use regularization instead of using PCA to reduce the dimension of the data. And the reason is, if
* 8:11
* you think about how PCA works, it does not use the labels y. You are just looking at your inputs xi, and you're using that to find a lower-dimensional approximation to your data. So what PCA does, is it throws away some information.
* 8:26
* It throws away or reduces the dimension of your data without knowing what the values of y is, so this is probably okay using PCA this way is probably okay if, say 99 percent of the variance is retained, if you're keeping most of the variance, but it might also throw away some valuable information. And it turns out that if you're retaining 99% of the variance or 95% of the variance or whatever, it turns out that just using regularization will often give you at least as good a method for preventing over-fitting
* 8:58
* and regularization will often just work better, because when you are applying linear regression or logistic regression or some other method with regularization, well, this minimization problem actually knows what the values of y are, and so is less likely to throw away some valuable information, whereas PCA doesn't make use of the labels and is more likely to throw away valuable information. So, to summarize, it is a good use of PCA, if your main motivation to speed up your learning algorithm, but using PCA to prevent over-fitting, that is not a good use of PCA, and using regularization instead is really what many people would recommend doing instead. Finally, one last misuse of PCA. And so I should say PCA is a very useful algorithm, I often use it for the compression on the visualization purposes.
* 9:50
* But, what I sometimes see, is also people sometimes use PCA where it shouldn't be. So, here's a pretty common thing that I see, which is if someone is designing a machine-learning system, they may write down the plan like this: let's design a learning system. Get a training set and then, you know, what I'm going to do is run PCA, then train logistic regression and then test on my test data. So often at the very start of a project, someone will just write out a project plan than says lets do these four steps with PCA inside.
* 10:20
* Before writing down a project plan the incorporates PCA like this, one very good question to ask is, well, what if we were to just do the whole without using PCA. And often people do not consider this step before coming up with a complicated project plan and implementing PCA and so on. And sometime, and so specifically, what I often advise people is, before you implement PCA, I would first suggest that, you know, do whatever it is, take whatever it is you want to do and first consider doing it with your original raw data xi, and only if that doesn't do what you want, then implement PCA before using Zi.
* 11:01
* So, before using PCA you know, instead of reducing the dimension of the data, I would consider well, let's ditch this PCA step, and I would consider, let's just train my learning algorithm on my original data. Let's just use my original raw inputs xi, and I would recommend, instead of putting PCA into the algorithm, just try doing whatever it is you're doing with the xi first. And only if you have a reason to believe that doesn't work, so that only if your learning algorithm ends up running too slowly, or only if the memory requirement or the disk space requirement is too large, so you want to compress your representation, but if only using the xi doesn't work, only if you have evidence or strong reason to believe that using the xi won't work, then implement PCA and consider using the compressed representation.
* 11:47
* Because what I do see, is sometimes people start off with a project plan that incorporates PCA inside, and sometimes they, whatever they're doing will work just fine, even with out using PCA instead. So, just consider that as an alternative as well, before you go to spend a lot of time to get PCA in, figure out what k is and so on. So, that's it for PCA. Despite these last sets of comments, PCA is an incredibly useful algorithm, when you use it for the appropriate applications and I've actually used PCA pretty often and for me, I use it mostly to speed up the running time of my learning algorithms. But I think, just as common an application of PCA, is to use it to compress data, to reduce the memory or disk space requirements, or to use it to visualize data.
* 12:34
* And PCA is one of the most commonly used and one of the most powerful unsupervised learning algorithms. And with what you've learned in these videos, I think hopefully you'll be able to implement PCA and use them through all of these purposes as well.