Advice for Applying PCA

We can use PCA to speed up algorithm running time:

- We're going to see how
- and see a general advice

Supervised learning speedup

Let's say that we have a supervised learning problem (with inputs x and labels y):

$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$

- Let's sat that x is a 10 000 dimensional feature vector (high dimensional vector $x^{(m)} \in \mathbb{R}^{100}$)
  - e.g. We may have a computer vision problem where we have a lot of features say: 100 x 100 images = 10 000 pixels
  - Such a huge feature vector will make the algorithm slow
  - **With PCA we can reduce the dimensionality and make it tractable**

- How we do a learning algorithm more efficient with PCA?
  1. **Extract inputs:**
     - Unlabeled dataset: $x^{(1)}, x^{(2)}, \dots, x^{(m)} \in \mathbb{R}^{10000}$
  2. **Apply PCA to x vectors:**
     - So we now have a reduced dimensional feature vector z
       - $z^{(1)}, z^{(2)}, \dots, z^{(m)} \in \mathbb{R}^{1000}$
  3. **This gives us a new training set:**
     - Each vector can be re-associated with the label $(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \dots, (z^{(m)}, y^{(m)})$
  4. **Take the reduced dimensionality data set and feed to a learning algorithm with $h_\theta(z) = \frac{1}{1+e^{-\theta^T z}}$**
     - Use y as labels and z as feature vector

If we have a new example map from higher dimensionality vector to lower dimensionality vector $x \rightarrow z$, then feed into learning algorithm

Note: Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA **only on the tranining set**.

- The mapping computes a set of parameters
  - Feature scaling values or U_{reduce}
- Parameter learned by PCA
  - Should be obtained only by determining PCA on your training set
- **So we use those learned parameters for our cross validation data and test set:**
  - Concretely, this mapping can be applied as well to the examples $x_{CV}^{(i)}$ and $x_{\text{test}}^{(i)}$ in the cross validation and test sets.

Just to summarize, when we're running PCA, run our PCA **only on the training set portion of the data not the cross-validation set or the test set portion of our data**. And that defines the mapping from $x^{(i)}$ to $z^{(i)}$ and we can then apply that mapping to our cross-validation set and our test set.

In this example we talked about reducing the data from 10,000 dimensional to 1,000 dimensional and still retain most of the variance and we can do this barely effecting the performance, in terms of classification accuracy, let's say barely affecting the classification accuracy of the learning algorithm.

This is actually not that unrealistic, for many problems we actually reduce the dimensional data.

Application of PCA

- **Compression**
  - Reduce memory/disk needed to store data
  - Speed up learning algorithm
    - Choose k by 95% or 99% of variance retained
- **Visualization**
  - Typically chose k = 2 or k = 3
    - Because we can plot only 2D and 3D data sets

There is often one frequent misuse of PCA

Bad applications of PCA

Bad use of PCA: To prevent overfitting

- Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$.
- Thus, fewer features, less likely to overfit.

Here's the reasoning behind this method:

- If we have $x^{(i)}$ we have n features, $z^{(i)}$ has k features which can be lower
- If we only have k features (say $k < n$, 1,000 < 10,000) then maybe we're less likely to over fit...
- **Some people think of PCA as a way to prevent over-fitting**.

We can't use PCA to prevent overfitting, doesn't work (bad application):

This might work OK, but isn't a good way to address overfitting. Use regularization instead.

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

If we think about how PCA works, it does not use the labels y, we are just looking at our inputs $x^{(i)}$, and we're using that to find a lower-dimensional approximation to our data. So what PCA does is **it throws away some information**

- Probably OK if we're keeping most of the data
  - So we have to go to like 95-99% variance retained
  - But it might also throw away some valuable information
  - So here regularization will give us **at least** as good a way to solve over-fitting

So, to summarize, it is a good use of PCA, if our main motivation is to **speed up our 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.

PCA is sometimes used where it shouldn't be

Design of ML system:

- Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
- Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$
- Traing logistic regression on $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$
- Test on test set: Map $x_{\text{test}}^{(i)}$ to $z_{\text{test}}^{(i)}$. Run $h_\theta(z)$ on:
  - $\{(z_{\text{test}}^{(1)}, y_{\text{test}}^{(1)}), \dots, (z_{\text{test}}^{(m)}, y_{\text{test}}^{(m)})\}$

How about doing the whole thig without using PCA?

Before implementing PCA, first try running whatever we want to do with the original/raw data $x^{(i)}$. Only if that doesn't do what we want, then implementing PCA and consider using $z^{(i)}$.

Concretely only if we have a reason to believe PCA will help should we then add PCA:

- PCA is easy enough to add on as a processing step
- Try without PCA first

Video Question: Which of the following are good / recommended applications of PCA? Select all that apply.

> To compress the data so it takes up less computer memory / disk space

> To reduce the dimension of the input data so as to speed up a learning algorithm

- Instead of using regularization, use PCA to reduce the number of features to reduce overfitting

> To visualize high-dimensional data (by choosing k = 2 or k = 3)