CSE 252A: Deep Networks I

Lecturer: David Kriegman

Scribed by Owen Jow on November 29, 2018

1 Eigenfaces vs. Fisherfaces

In the case of classification, we would like to reduce the dimensionality of our feature space in a way that leaves only distinct, independent feature dimensions.

1.1 PCA

- We can do this via PCA by finding the directions of greatest spread (a basis), and constructing/applying a matrix which represents points from the original space in the new basis.
- PCA finds the lower-dimensional space that best approximates the data in an SSD sense.
- If the number of samples is much less than the original dimensionality, use SVD for PCA.
- One problem with PCA is that lost principal components might include small details that are important for e.g. recognition, like a small marking which doesn't appear on most of the data.

1.2 Fisher's Linear Discriminant

If we care about facial recognition in different lighting (which we should), it might be the case that the first principal components capture variation more dependent on lighting, and the later principal components capture variation less dependent on lighting.

But to be sure, we should use Fisher's linear discriminant (FLD) to get a basis which focuses on variations which are less lighting-dependent. This method uses two covariance matrices: one for the between-class spread, and one for the within-class spread. In other words, we take advantage of the supervised nature of our task to construct the subspace (by contrast, PCA operates unsupervised).

PCA maximizes scatter. FLD maximizes a ratio which is large when the numerator (*projected between-class spread*) is large and/or the denominator (*projected within-class spread*) is small. Basically, it clusters things so that the recognition task is easy.

Also called *Fisher's linear discriminant analysis (LDA)*.

2 Feature Extraction

Before the explosion of deep networks, we might've had a long and hand-engineered feature extraction process that went something like (1) preprocess (e.g. normalize) the image, (2) apply a filter bank

(e.g. Gabor) to each color channel, (3) take max response within eight groups of filters to get an 8D vector for each color channel (24D vector total), (4) convert these into bag-of-words representations, and (5) histogram the words over image regions at different scales.

Then we might have passed the feature vectors to an SVM classifier.

A neural network will do all of this automatically.

3 Neural Networks

A single perceptron is $\mathbf{w}^T\mathbf{x} + b$. We can represent `OR` and `AND` but not `XOR`. With two or more layers, we can represent `XOR` (and any boolean function). Then, with sufficient neurons and nonlinearities (sigmoids, hyperbolic tangents, ReLUs), we can represent any function (*universal approximation*).

So representational capacity is high. But training is another matter. For training, it has been shown that *deep* networks are more efficient in that they require less hidden units than shallower networks to represent the same functions. Stack more layers!

To train, we use gradient descent[1][2] with some loss function.

Neural networks are just learned composite functions $l_n(\cdots l_3(l_2(l_1(\mathbf{x}))))$. They are trained to approximate functions, i.e. they are function approximators.

[1] Gradient gives direction in input space that makes the output increase the most. If you change input infinitesimally in the opposite direction, output will decrease. This is gradient descent (except steps are finite in GD).

[2] Gradient descent is nice because we only deal with first derivatives (linear in number of parameters) instead of having to take the Hessian or something (like many other numerical optimization methods).