CS 189: Introduction to Machine Learning - Discussion 13

1. Kernel k-means

   Suppose we have a dataset $\{x_i\}_{i=1}^N, x_i \in \mathbb{R}^n$ that we want to split into K clusters. Furthermore, suppose we know a priori that this data is best clustered in a large feature space $\mathbb{R}^m$, and that we have a feature map $\phi : \mathbb{R}^n \to \mathbb{R}^m$. How should we perform clustering in this space?

   (a) Write the objective for K-means clustering in the feature space (using the squared L_2 norm in the feature space). Do so by explicitly constructing cluster centers $\{\mu_k\}_{k=1}^K$ with all $\mu_k \in \mathbb{R}^m$.

Solution:

$$L = \sum_{k=1}^{K} \sum_{x_i \in S_k} \|\phi(x_i) - \mu_k\|^2$$

(b) Write an algorithm that minimizes the objective in (a).

Solution:

1. Compute $\phi(x_i)$ for every point x_i.
2. Do the standard k-means on $\{\phi(x_i)\}$.

(c) Write an algorithm that minimizes the objective in (a) without explicitly constructing the cluster centers $\{\mu_k\}$. Assume you are given a kernel function $\kappa(x, y) = \phi(x) \cdot \phi(y)$.

Solution:

We proceed by coordinate descent on the objective in (a). First, given a clustering, the setting of μ_i that minimizes L is

$$\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} \phi(x)$$

Second, given a setting of the μ's, the optimal clustering is given by assigning x_i to the cluster $\arg\min_{1 \le k \le K} f(i, k)$, where

$$f(i, k) = \|\phi(x_i) - \mu_k\|^2$$

To kernelize this, we write

$$f(i,k) = \phi(x_i)\cdot\phi(x_i) - 2\phi(x_i)\cdot\mu_k + \mu_k\cdot\mu_k$$

Substituting the setting of μ_k,

$$= \phi(x_i)\cdot\phi(x_i) - \frac{2}{|S_k|}\sum_{x_j\in S_k}\phi(x_i)\cdot\phi(x_j) + \frac{1}{|S_k|^2}\sum_{x_j,x_l\in S_k\times S_k}\phi(x_j)\cdot\phi(x_l)$$

Now we can replace the inner products with kernel evaluations

$$= \kappa(x_i,x_i) - \frac{2}{|S_k|}\sum_{x_j\in S_k}\kappa(x_i,x_j) + \frac{1}{|S_k|^2}\sum_{x_j,x_l\in S_k}\kappa(x_j,x_l)$$

This yields the following algorithm:

1. Compute the kernel matrix $G_{ij} = \kappa(x_i,x_j)$.
2. Start with an initial clustering $\{S_k\}$.
3. Compute the new cluster index for each x_i as $\arg\min_{1\le k\le K} f(i,k)$.
4. Update $\{S_k\}$
5. Repeat steps (3) and (4) until convergence.

2. Parameters in Fully Connected Neural Networks

(a) The neural network you built for Homework 6 had 784, 200, and 10 units in each layer. Determine the total number of parameters in this neural network. Be sure to include bias terms.

Solution: $785\cdot 200 + 201\cdot 10 = 159010$

(b) You would like to classify larger grayscale images (size 224×224). You have 1000 output classes and 3 hidden layers with 6000 units each. Approximately how many parameters are in this network?

Solution: $50000\cdot 6000 + 6000\cdot 6000 + 6000\cdot 6000 + 6000\cdot 1000 = 378$ million

3. Convolutional Neural Networks

A typical ConvNet has a series of convolutional and pooling layers, followed by some fully connected layers and an output layer. You already know how to implement fully connected layers. This question will give you some intuition about convolutional and pooling layers.

(a) A typical **convolutional layer** might consist of 128 3×3 filters. During the forward pass, we slide each filter across the width and height of the input layer and compute the dot product between the filter and the part of the image "underneath" the filter. So if our input image is 200×200 grayscale, we would end up with a 200×200 activation map for each of our 128 filters. Let's try it on the following example. The image is zero-padded for convenience; your output should be 3×3.

Input image: $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 1 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

Filter: $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

Solution: $\begin{bmatrix} 3 & 1 & 2 \\ -1 & 4 & 1 \\ 2 & 2 & -3 \end{bmatrix}$

This is tedious to do by hand; you should probably only do the first row.

Also, this is actually cross-correlation, but it's convolution if you just flip the filter horizontally and vertically.

(b) **Pooling layers** are usually inserted between convolutional layers to reduce the dimensions of the image. This helps speed up computation of the network and increases the receptive field. A commonly used pooling layer is max-pooling with kernel size 2 and stride 2, which means that the max is taken over every block of 4 pixels. Try performing max-pooling on the following input:

Input image: $\begin{bmatrix} 2 & 9 & 3 & 8 \\ 1 & 2 & 4 & 8 \\ 2 & 4 & 5 & 1 \\ 0 & 3 & 2 & 6 \end{bmatrix}$

Solution: $\begin{bmatrix} 9 & 8 \\ 4 & 6 \end{bmatrix}$

(c) **Receptive fields** The receptive field refers to how much of the original image a layer "sees." The very first conv layer with kernel size 3×3 has a receptive field of 3×3. What is the receptive field of a second 3×3 conv layer that immediately follows the first conv layer? What is the receptive field of a second 3×3 conv layer if a 2×2, stride 2 max-pooling layer is inserted in between?

Solution: The receptive field of the 2nd conv layer is 5×5.

If there is a max-pooling layer in the middle, the RF of the 2nd conv layer is 8×8. We use the following equations:

$RFsize = PrevRFSize + PrevRFStride \times (KernelSize - 1)$

$RFstride = PrevRFStride \times Stride$

These details are not important; the main takeaway is that pooling layers increase the receptive field and cut down on spatial resolution.

(d) **Layer design** Would you prefer to stack 3 convolutional layers with kernel size 3×3, or use 1 convolutional layer with kernel size 7×7? We use ReLUs after each convolutional layer.

Solution: Although both designs have the same receptive field size, the 3 stacked layers are more expressive because the network can build up more complex features over many layers, while the 7×7 conv layer has only 1 non-linearity to work with. The stacked design also has fewer parameters: $27x$ vs $49x$ where x is the number of filters.

(e) **Calculating Parameters** Your ConvNet architecture is INPUT-(CONV-POOL)*5-FC-FC-OUTPUT. Input images are grayscale, 224×224. All CONV layers use 128 3×3 filters, with 1 pixel per side padding. All POOL layers are 2×2 stride 2 max-pooling. Both FC layers have 4096 units. Approximately how many parameters are in this network?

Solution: Pooling layers don't have parameters.

There are 5 conv layers. The first conv layer has $128 \cdot 3 \cdot 3$ parameters, and the remaining 4 conv layers have $128^2 \cdot 3 \cdot 3$ parameters. This is because the extent of the connectivity along the depth axis is always equal to the depth of the input (128). So the conv layers contribute 148608 parameters.

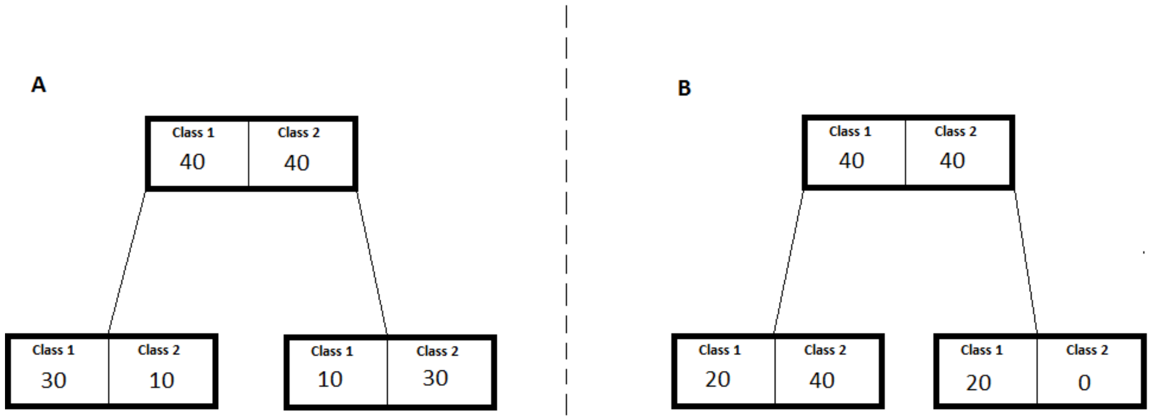
We need to calculate the spatial resolution of the image immediately before the first FC layer. The conv layers don't change the spatial resolution. The image starts at 224×224 and undergoes 5 pooling operations, so the ending resolution is $224/(2^5) \times 224/(2^5) = 7 \times 7$. Recall that there are still 128 filter maps.

So for all the FC layers, we have: $128 \cdot 7 \cdot 7 \cdot 4096 + 4096 \cdot 4096 + 4096 \cdot 1000 \approx 46$ million.

As we can see, the convolutional neural network is deeper than the fully connected network in 2(b) but uses fewer parameters!

4. Decision Tree Review

(a) Let's say we are trying to build a 2-class decision tree and are currently at a node with 40 samples from each class. Our two possible splits are illustrated below. If we are trying to minimize misclassification error, which split should we select? What if we are trying to minimize entropy?



Solution: Using misclassification error as a heuristic will favor split A, while using entropy as a heuristic will favor split B.

Misclassification error:

Split A will result in a misclassification error of $\frac{40}{80}\frac{10}{40} + \frac{40}{80}\frac{10}{40} = .25$

Split B will result in a misclassification error of $\frac{60}{80}\frac{20}{60} + \frac{20}{80}\frac{1}{20} = .2625$

Entropy:

Split A will result in a weighted entropy of $\frac{40}{80}(-\frac{30}{40}\log_2\frac{30}{40} - \frac{10}{40}\log_2\frac{10}{40}) + \frac{40}{80}(-\frac{30}{40}\log_2\frac{30}{40} - \frac{10}{40}\log_2\frac{10}{40}) = .81$

Split B will result in a weighted entropy of $\frac{60}{80}(-\frac{20}{60}\log_2\frac{20}{60} - \frac{40}{60}\log_2\frac{40}{60}) + \frac{20}{80}(-\frac{19}{20}\log_2\frac{19}{20} - \frac{1}{20}\log_2\frac{1}{20}) = .76$

Therefore, if you are trying to minimize entropy, you will select split B, but if you are trying to minimize misclassification error, you will select split A.

(b) An optimal decision tree is one that gives the most accurate classification with the least depth. Does using entropy as a heuristic always guarantee you will have an optimal decision tree?

Solution: No, entropy is a greedy heuristic and will not necessarily find the global optimum. Finding the optimal decision tree is an NP-complete problem.