Chapter 1

Lecture 5

1.1 Segmented Least Squares

Let's say we have a noisy graph of data that we would like to generalize. We can use a line of best fit given a bunch of points $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$ where $x_1 \leq x_2 \leq ... \leq x_n$.
There are a lot of formulas to find the error of a line of best fit but we can square the distances between every point and hte line and sum them.

Definition: Error

$$\sum_{i=1}^{n}(y_i - f(x_1))^2 = \sum(y_2 - ax_i - b)^2$$

We can find this line of minimum error using a formula. We won't be proving this or going over how it works in this class.

Definition: Line of Minimum Error

$$a = \frac{n\sum_i x_i y_i - (\sum x_i)(\sum y_i)}{n\sum x_i^2 - (\sum x_i)^2}$$

$$b = \frac{\sum y_i - a\sum x_i}{n}$$

We don't have to know these formulas, but the most important thing to know is that it takes $O(n)$ to comput this line.

So now we can generate a line of best fit for any arbituary graph. But what if our data looks nonlinear? There are a lot of ways to try to fix this. We could try to fit a polynomial or exponential function.
This problem will focus on a different approach. We could try to fit the graph in multiple line segments of best fit instead of just one.

Problem: Segmented Least Squares

Given: A set of data points
Goal: Fit a bunch of line segments to minimize squared error

If we could make as many line segments as we wanted, we could get zero error by just connecting all the points to adjacent points with n line segments. So obviously we need a way to somehow limit the number of line segments.

Want: a small number of segments and small error
One approach could be to fix one variable and minimize the other variable. In this solution we will try to combine the objectives together.

Objective: to minimize total error + c*number of line segments

If c is large, we are trying to make sure there are as few line segments. If c is small, we can use more segments for better error.

Claim

If we know the breakpoints between the lines, then we can easily compute the best lines to minimize the error.

How? We can just use the formula after the breakpoints to find the best lines.
Note: The lines do not have to be connected in our example. They can jump around and such.

1.2 Algorithm Time

Given: points $p_1...p_n$
$OPT(j)$ = value of best solution given only points $p_1...p_j$

Simple Observation: The breakpoint where the last segment starts must be at some p_i.

$OPT[n]$ = best of {best if last break is at p_1, best if last break is at p_2, ... best if last break is at p_n }
e_{ij} = error of best line fitting points $p_i...p_j$
$OPT[n] = e_{in} + OPT[i-1] + c$ for some i
$OPT[j] = \min_{i \leq i \leq j}\{e_{ij} + C + OPT(i-1)\}$

1.3 Solution

Base Case: $OPT[0] = 0$
Fill in the array from left to right
Final Answer: $OPT[n]$
That is the extent of what needs to be written on homeworks/tests but here's the psuedocode for sake of completion.

```
For all i <= j:
    compute e[i][j]
OPT[0] = 0
For j = 1 to n:
    OPT[j] = recurrence
return OPT[n]

if j = 0: return null
else
    Find i minimizing e[i][j] + C + OPT[i-1]
    return best line for [p[i] ... p[j]] + find segment(i-1)
```

Runtime: $O(n)$
We can compute the error with dynamic programming as well.

1.4 Weighted Interval Scheduling

We can apply the same logic that we used when coming up with the simple observation for segmented least squares to weighted interval scheduling.

Observation: In OPT, the last interval that is selected must be somewhere.

$$OPT(i) = max_{i \leq j}\{w_i + OPT[p_i]\}$$

The point being there are many ways to look at a problem and come up with a DP solution. In this case, both work, but a naively implemented solution would have this new reccurence run at a runtime of $O(n^2)$ as compared to our original $O(n)$.