Lecture #21

- So far: kNN classifier: basically local averaging of nearby labels — no long range connections in data
  - SVM: parametric separation between classes sought, perhaps after kernelization
- A different approach begins with another view on classification. We wish to learn, for a data point X_i, the associated label y_i.
- So, we can think we are trying to learn a function $g: \mathbb{R}^D \to \{-1, 1\}$ s.t. $g(x_i) =$ label of point x_i.
- Supervised classification: Given training examples $\{(x_i, y_i)\}_{i=1}^n$, build $g: \mathbb{R}^D \to \{-1, 1\}$
- How should we evaluate g?
- First idea: g should do well on the labeled points $\{(x_i, y_i)\}_{i=1}^n$.

  Problem: If we just have g perform well on the training set, no reason to think it will do well on new points ... need g to

do more than just memorize the training data — it needs to generalize well!

- How to construct such a y? SVM tries to ensure good generalization by maximizing the hyperplane margin.
- We need to balance how well we do on training with generalization to unseen points — bias variance tradeoff.

- Neural networks provide a method for learning $y(x)$ that is highly flexible. ~~We try to fit a multivariate [illegible]~~
- We think of $y(x)$ as a function that depends on some parameters w: $y(x,w) = f\left(\sum_{j=1}^{M} w_j \phi_j(x)\right)$. We can think f is a fixed nonlinear function and $\phi_j(x)$ are some basis functions, also fixed.
- Our goal is to choose w that causes $y(x,w)$ to predict new points well. This is typically done (at least as a first attempt) by

Choosing w that minimize the error on the training set:

$$w^* = \arg\min_w \underbrace{\sum_{i=1}^{n} |y(x_i, w) - y_i|}_{\text{minimize labeling error.}} \quad (\star)$$

- Very flexible approach, but with hazards: 1.) How to pick M?
  Too small, may not be able to fit the training data well!
  Too big, may not generalize well.
  2.) How to solve $(\star)$?

- Neural networks make things even more complicated by having ϕ_i depend on a parameter!

- Let $x = (x_1, \ldots, x_D)$. Let $a_j = \sum_{i=1}^{D} \underbrace{w_{ji}^{(1)}}_{\text{weights}} x_i + \overbrace{w_{j0}^{(1)}}^{\text{biases}}$ be a linear combination of the variables of x.

- These combinations are the run through a (non-linear) activation function

$$h(a_j) := z_j = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right).$$

- We can then re-combine the z_j!

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

- Let σ be an output activation function, i.e. $\sigma(a) = \frac{1}{1+e^{-a}}$.

- Then we can write an estimator for $y(x)$ as

$$y_k(x, W) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right),$$

where $W =$ [illegible] all parameters, i.e. $\{w_{kj}^{(2)}\}$ [illegible]

- For classification, we want $y_1(x_i, W)$ big if $y(x_i) = -1$

$y_2(x_i, W)$ big if $y(x_i) = 1$ ←

- In principal, all I have to do is choose W so that this is likely to happen.

- Problem: very hard to set W! Lots of parameters! Lots of nonconvex optimization! Somehow, it can work...