Statistical Techniques in Robotics (16-831, F10) Lecture#15 (Wednesday November 2)

Boosting: Gradient Descent in Function Space

Lecturer: Alex Grubb *Scribe: Samantha Horvath* [1]

1 Gradient Descent in Function Space

Gradient descent for linear classifiers such as SVM can generalized for nonlinear classification in function space. Instead of minimizing over the space of possible weight vectors (or other applicable parameters), we minimize over over the space of possible functions. A linear minimzation (like for SVM):

$$\min_w \sum_{n=1}^{N} \max(0, 1 - y_n w^T x_n)$$

in function space becomes...

$$\min_f \sum_{n=1}^{N} \max(0, 1 - y_n f(x_n))$$

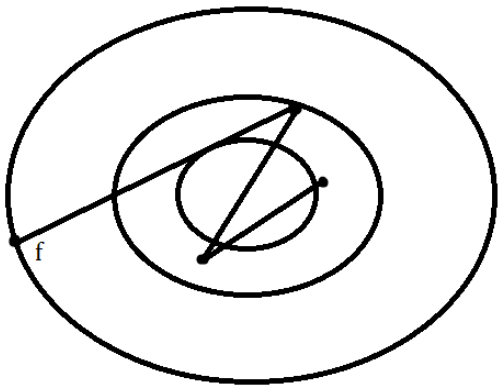

Figure 1: Gradient descent in function space: each point is a function

For gradient descent optimization in function space, we want to minimize the loss function:

$$\min_{f \in F} R[f]$$

$$\forall f : X \to \mathbb{R}$$

[1] Some content adapted from previous scribes: :(

Our general loss function is:

$$R[f] = \frac{1}{N}\sum_{n=1}^{N} l_n(f(x_n)) \tag{1}$$

1.1 L^2 Function Space

To do this, we will look at the L^2 function space, which is the set of all f for which the following function is bounded:

$$\int_X |f(x)|^2 p(x)dx \tag{2}$$

This includes most functions. For functions in the L^2 there is a natural inner product:

$$< f, g >= \int_X f(x)g(x)p(x)dx = E_X[fg] \tag{3}$$

and induced norm:

$$||f||^2 = \int_X |f(x)|^2 p(x)dx = E_X[f^2] \tag{4}$$

These are defined over all the data in X. Typically, we don't have all of the data. We can approximate the inner product with:

$$< f, g >= \frac{1}{N}\sum_{n=1}^{N} f(x_n)g(x_n) = E_{\hat{X}}[fg] \tag{5}$$

1.2 Subgradients in Function space

The definition of the subgradient in function space is:

$$R[g] \geq R[f] + < \nabla R[f], g - f > \tag{6}$$

Note that it is very similar to the definition of hte subgradient of a function. We will now plug in out loss functiong given in (1) and definition of inner product given in (5) to derive a representation of the gradient

$$\frac{1}{N}\sum_{n=1}^{N} l_n(g(x_n)) \geq \frac{1}{N}\sum_{n=1}^{N} l_n(f(x_n)) + \frac{1}{N}\sum_{n=1}^{N} \nabla R[f](x_n)[g(x_n) - f(x_n)]$$

$$0 \geq \sum_{n=1}^{N} [l_n(f(x_n)) + \nabla R[f](x_n)[g(x_n) - f(x_n)] - l_n(g(x_n))]$$

Here will will require each element of the sum to be less than zero:

$$0 \geq l_n(f(x_n)) + \nabla R[f](x_n)[g(x_n) - f(x_n)] - l_n(g(x_n))$$

Rearranging...

$$l_n(g(x_n)) \geq l_n(f(x_n)) + \nabla R[f](x_n)[g(x_n) - f(x_n)] \tag{7}$$

By inspection, we can see that the sudgradient of the loss function in function space is equivalent to the subgradient of the pointwise loss function at each x_n.

$$\nabla R[f](x) = \begin{cases} \delta l_n(f(x_n)) & \text{if } x = x_n \\ undef & \text{if } otherwise \end{cases} \tag{8}$$

So for the squared-error loss function:

$$R[f] = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{2}(f(x_n) - y_n)^2 \tag{9}$$

the subgradient would be:

$$\nabla R[f](x_n) = (f(x_n) - y_n) \tag{10}$$

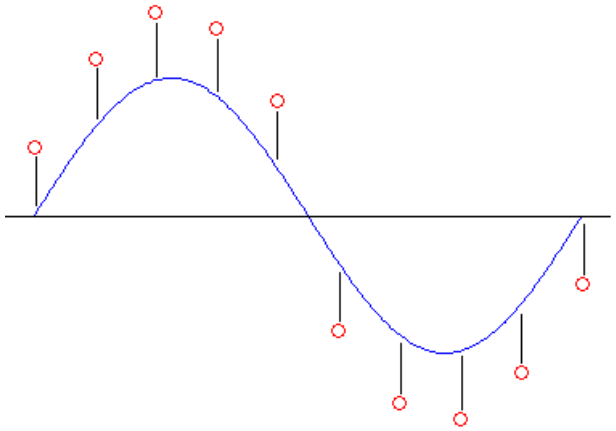

Figure 2: The subgradient of our loss function(indicated by the black lines) is the difference between the observed(red) and predicted(blue) values at each x_n

1.3 Functional Gradient Descent, Take 1

1. compute $\nabla_t = \nabla R[f_t]$
2. Select step size α_t
3. update: $f_{t+1} \leftarrow f_t + \alpha_t \nabla_t$

Problem - the function is undefined at all points except the training points. In the case of our squared-error loss function, it completely overfits to the training data.

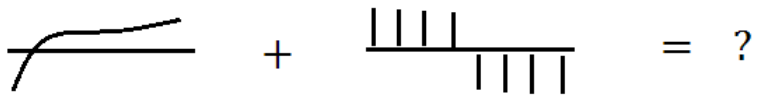

Figure 3: The update rule results in the new function being undefined at many points

Solution- find a continuously defined function h that closely matches the gradient. We will select h from a group of candidate functions H

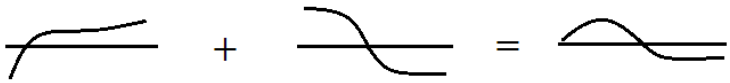

Figure 4: Find a continuous function that closely matches the gradient

1.4 Choosing a Projection

How to chose a projection h? - project the gradient onto all of the candidate functions in vector space, and choose the "closest" one

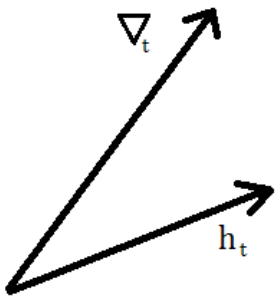

Figure 5: The gradient and candidate function in vector space

We can minimize distance:

$$h_t = \underset{h \in H}{\operatorname{argmin}} ||h - \nabla_t||^2 \tag{11}$$

Or, we can minimize angle (maximize projection):

$$h_t = \underset{h \in H}{\text{argmax}} \frac{< \nabla_t, h >}{||h||} \tag{12}$$

Minimizing the angle is equivalent to weighted classification for certain problems.

1.5 Functional Gradient Descent, Take 2

Now we want to solve:

$$\min_{f \in F} R[f]$$

such that:

$$f = \sum_T \alpha_t h_t$$

$$\forall t : h_t \in H, \alpha_t \in \mathbb{R}$$

1. compute $\nabla_t = \nabla R[f_t]$
2. Project using $h_t = \text{argmin}_{h \in H} ||h - \nabla_t||^2$ or $h_t = \text{argmax}_{h \in H} \frac{<\nabla_t, h>}{||h||}$
3. Select step size α_t
4. update: $f_{t+1} \leftarrow f_t + \alpha_t h_t$

How should we determine alpha?

Line search (i.e. the "ideal" step):

$$\alpha_t = \underset{\alpha}{\text{argmin}} R[f + \alpha h_t] \tag{13}$$

Projected step (move some fraction of the projection):

$$\alpha_t = -\eta_t \frac{< \nabla_t, h >}{||h||^2} \tag{14}$$

... Or just use a fixed step size. The fixed step size can help to reduce overfitting caused by moving too far along the chosen projection. A fixed step size is often useful for noisy data.

2 Extensions: Boosting and AdaBoost

2.1 The Boosting Algorithm

Given training data, repeat until converged:

1. Create a temporary dataset based on current errors (compute gradient)
2. Train a weak learner on the error dataset (Project onto candidate functions)
3. Additively incorporate the new weak learner (Follow projection for some step size)

As you can see, functional gradient descent is the generic approach to boosting.

2.2 AdaBoost

AdaBoost is a very popular boosting algorithm. It uses an exponential loss function, which is a smooth upper bound on zero-one loss:

$$R[f] = \frac{1}{N}\sum_{n=1}^{N} \exp((-y_n f(x_n)) \tag{15}$$

where:

$$y_n : \{+1, -1\}$$

The subgradient for this loss function is:

$$\nabla R[f](x) = \begin{cases} -y_n \exp(-y_n f(x_n)) & \text{if } x = x_n \\ undef & \text{if } otherwise \end{cases}$$

This can be split into weights:

$$w_{nt} = \exp(-y_n f(x_n))$$

and labels:

$$y_{nt} = -y_n$$

So, the AdaBoost algorithm is:

1. compute $\nabla_t = \nabla R[f_t]$
2. Project using $h_t = \text{argmax}_{h \in H} \frac{<\nabla_t, h>}{||h||}$
3. compute step size $\alpha_t = \text{argmin}_\alpha R[f + \alpha h_t]$
4. update: $f_{t+1} \leftarrow f_t + \alpha_t h_t$

For AdaBoost, α can be determined analytically:

$$\underset{\alpha}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} \exp[-y_n(f(x_n) + \alpha h_t(x_n))]$$

We now differentiate wrt α, and set equal to 0

$$= \sum_{n=1}^{N} -h_t(x_n) y_n \exp(-y_n(f(x_n)) \exp(-y_n \alpha h_t(x_n))$$

Now, we split the sums based upon the sign of y_n

$$0 = \sum_{h_t = y_n} -\exp(-y_n f(x_n)) \exp(-\alpha) + \sum_{h_t \neq y_n} \exp(-y_n f(x_n)) \exp(\alpha)$$

$$\exp(-\alpha) \sum_{h_t = y_n} \exp(-y_n f(x_n)) = \exp(\alpha) \sum_{h_t \neq y_n} \exp(-y_n f(x_n))$$

Taking the natural log of both sides...

$$-\alpha \ln[\sum_{h_t = y_n} \exp(-y_n f(x_n))] = \alpha \ln[\sum_{h_t \neq y_n} \exp(-y_n f(x_n))]$$

$$2\alpha = \ln[\sum_{h_t = y_n} \exp(-y_n f(x_n))] - \ln[\sum_{h_t \neq y_n} \exp(-y_n f(x_n))]$$

This is often written as:

$$\alpha = \frac{1}{2} \ln \frac{1 - \epsilon}{\epsilon} \tag{16}$$

where:

$$\epsilon = \sum_{n=1}^{N} w_n 1(h_t(x_n) \neq x_n) \tag{17}$$

AdaBoost has the property that marginally better weak learners lead to increasingly better training performance. So long as the weak learner at each iteration does at least slightly better than random, the overall performance will increase.