Lecture 11: Adaboost and the Viola-Jones Face Detector

Mark Hasegawa-Johnson
These slides are in the public domain

ECE 417: Multimedia Signal Processing, Fall 2023

Outline

Review: Images

An image is a signal, or a stack of signals. Often we write $I[c, m, n]$ where c is the color ($c \in \{1, 2, 3\}$), m is the row index, and n is the column index.

Review: Convolutional Neural Nets

1. Forward-prop is convolution:

$$Z[d,m,n] = W[d,c,m,n] * I[c,m,n]$$

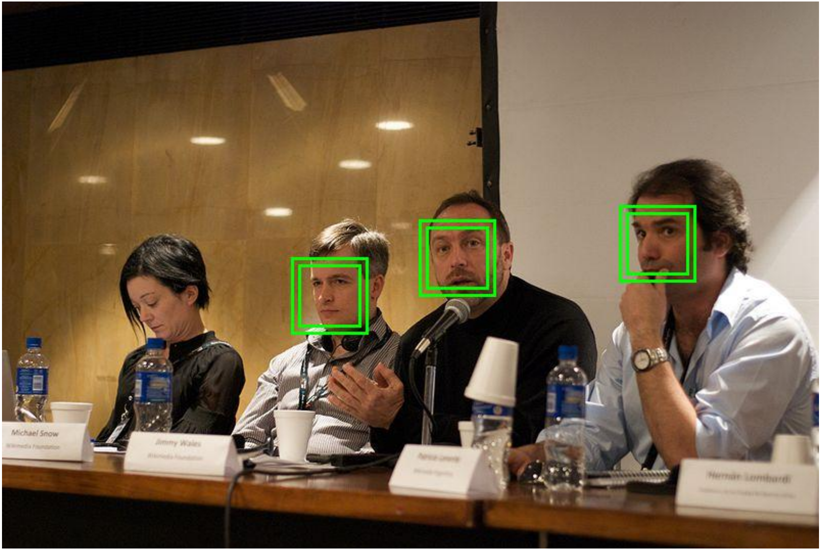
2. Back-prop is correlation:

$$\frac{\partial \mathcal{L}}{\partial I[c,m,n]} = W[d,c,m,n] \bigstar \frac{\partial \mathcal{L}}{\partial Z[d,m,n]}$$

3. Weight gradient is correlation:

$$\frac{\partial \mathcal{L}}{\partial W[d,c,m,n]} = \frac{\partial \mathcal{L}}{\partial Z[d,m,n]} \bigstar I[c,m,n]$$

Outline



https://commons.wikimedia.org/wiki/File:
Face_detection.jpg

Face Detection: Problem Definition

```
for m in range(M):
  for n in range(N):
    for height in range(number_rows - row):
      for width in range(number_cols - col):
        does (m,n,height,width) contain a face?
```

Why is Face Detection Difficult?

A CNN face detector might detect a face of width w and height h by training a "face detector" filter, $f[m,n]$ of width w and height h, then filtering the whole image to find the (m,n) where the face is located:

$$Z_{w,h}[m,n] = f_{w,h}[m,n] * I[m,n]$$

If the image is $M \times N$, this operation requires $w \times h \times M \times N$ multiplications.

Faces Come in Many Different Sizes

https://commons.wikimedia.org/wiki/File:Fraunhofer_-_Face_Detection_-_4406340595.jpg

Faces Come in Many Different Sizes

- Suppose the face width can be any size between $1 \leq w \leq W$
- Suppose the face height can be any size between $1 \leq h \leq H$
- Then we need WH different filters, $f_{w,h}[m,n]$, so that we can detect all the different faces
- Total computational complexity is:

$$\sum_{w=1}^{W}\sum_{h=1}^{H} whMN = \frac{1}{4}(W+1)^2(H+1)^2 MN \text{ multiplications/image}$$

Outline

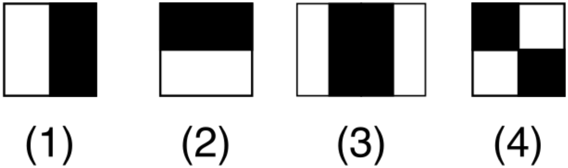
Haar-Like Features

Viola and Jones (2004) proposed solving the computational complexity problem by using very simple filters that they called "Haar-like features," because they resemble Haar wavelets.

1. Haar-like features require no multiplications, because for all n, $f[n]$ is either -1 or $+1$.
2. Haar-like features also require very few additions, because of a neat trick called the "integral image."

(1) Haar-like features require no multiplication

Haar-like features are convolutions, $Z[m,n] = f[m,n] * I[m,n]$, but the filters are $f[m,n] \in \{-1,1\}$. Shown below are 2-rectangle, 3-rectangle, and 4-rectangle filters. The black pixels are $f[m,n] = +1$, the white pixels are $f[m,n] = -1$.



https://commons.wikimedia.org/wiki/File:VJ_featureTypes.svg

(2) Haar-like features require few additions

Haar-like feature require very few additions, because they take advantage of an intermediate computation called the **integral image:**

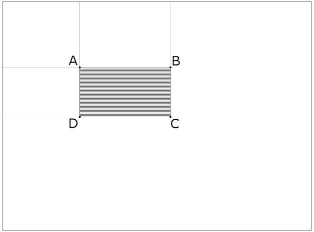
$$II[m,n] = \sum_{m'=1}^{m} \sum_{n'=1}^{n} I[m',n'], \quad 1 \le m \le M, 1 \le n \le N$$

The integral image is computed just once, for the entire image.

Summing a rectangle: Three additions

Using the integral image, the sum of all pixels inside a rectangle can be computed with only three additions.

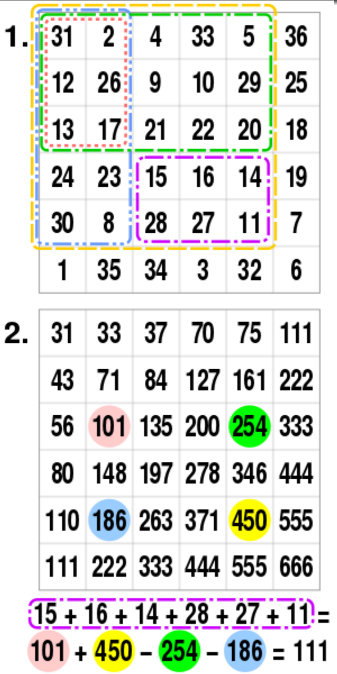
$$\sum_{m'=m}^{m+h}\sum_{n'=n}^{n+w} I[m',n'] = II[m+h,n+w] - II[m,n+w] - II[m+h,n] + II[m,n]$$



https://commons.wikimedia.org/wiki/File:Prm_VJ_fig3_computeRectangleWithAlpha.png

$$\sum_{m'=m}^{m+h} \sum_{n'=n}^{n+w} I[m', n']$$
$$= II[m+h, n+w] - II[m, n+w]$$
$$- II[m+h, n] + II[m, n]$$

Figure: `https://commons.wikimedia.org/wiki/File:Integral_image_application_example.svg`



The Training Token

Suppose we have a particular training datum x, which is an image from the training database, I, and a corresponding rectangle specifier (r_1, r_2, r_3, r_4) =(horizontal,vertical,width,height):

$$x = \begin{bmatrix} I[r_2, r_1] & \cdots & I_i[r_2, r_1 + r_3] \\ \vdots & \ddots & \vdots \\ I_i[r_2 + r_4, r_1] & \cdots & I_i[r_4 + r_2, r_1 + r_3] \end{bmatrix}$$

What are the different features we can compute from this image?

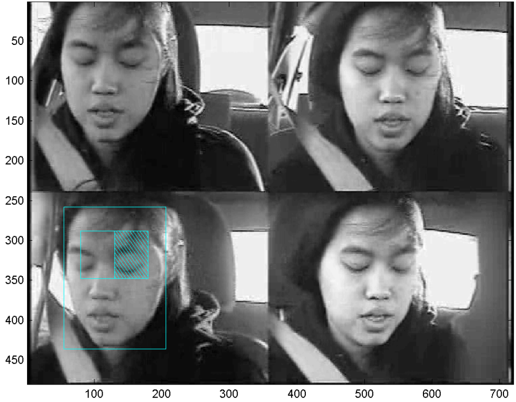
The Features

Viola & Jones define Haar-like features specified by:

- (ϕ_i, ϕ_2): Upper-left-corner of the sub-rectangle within the face rectangle, expressed as a fraction of the face rectangle, i.e., the upper left corner is $[m, n] = (r_2 + r_4\phi_2, r_1 + r_3\phi_1)$
- (ϕ_3, ϕ_4): Size of the sub-rectangle, expressed as a fraction of the face rectangle size, i.e., width in pixels is $r_3\phi_3$, height in pixels is $r_4\phi_4$
- (o_1, o_2): number of blocks in the horizontal and vertical directions, respectively.

The Features

For example, here is a feature specified by
$f = (\phi_1, \phi_2, \phi_3, \phi_4, o_1, o_2) = \left(\frac{1}{6}, \frac{1}{6}, \frac{2}{3}, \frac{1}{3}, 2, 1\right)$:



Outline

The Weak Classifier

Viola & Jones define the j^{th} "weak classifier" in terms of a feature $f_j(\cdot)$, a sign $p_j \in \{-1, 1\}$, and a threshold $\theta_j \in \Re$:

$$h_j(x) = \begin{cases} 1 & p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

The Weak Classifier: Complete Specification

Putting it all together, a "weak classifier," f_j, is specified by 8 numbers:

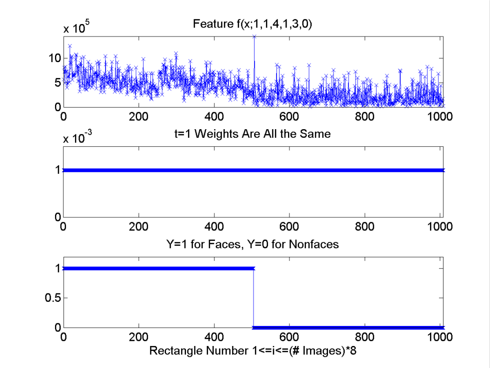
- $(\phi_{j,1}, \phi_{j,2}, \phi_{j,3}, \phi_{j,4})$: position of the subrectangle within the candidate face rectangle
- $(o_{j,1}, o_{j,2})$: number of blocks within the subrectangle
- θ_j: threshold above or below which we should detect a face
- p_j: sign of the weak classifier: are faces detected by being below (+1) or above (-1) the threshold feature value?

Selecting the Weak Classifier

- How do we choose $(\phi_1, \phi_2, \phi_3, \phi_4, o_1, o_2, p, \theta)$?
- To start with, let's suppose that we are evaluating a candidate feature, $f_j = (\phi_{j,1}, \phi_{j,2}, \phi_{j,3}, \phi_{j,4}, o_{j,1}, o_{j,2})$.
- We want to find the values of p_j and θ_j that minimize the training corpus error rate for this f_j, and we want to calculate the value of that error rate.

Feature Values, Weights, and Labels

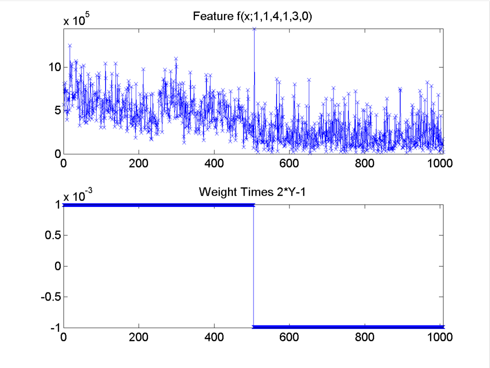
- For every training token x_i, find the feature value $f_j(x_i)$.
- Second, assign a weight to every token, $w_j(x_i)$. To start with, all the weights are equal, $w_j(x_i) = \frac{1}{n}$ where n is the number of tokens.
- Third, list the target labels, $y_i \in \{1, 0\}$.



Feature Values, Weights× Labels

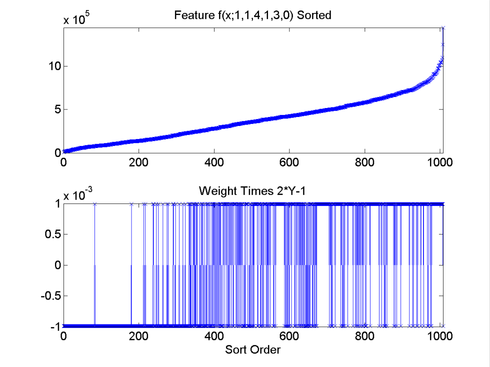
- Convert the labels from $\{0, 1\}$ to $\{-1, 1\}$
- Multiply each label times its weight, to give its "signed importance:"

$$s_i = w(x_i) \times (2y_i - 1)$$



Sorted Features, Argsorted Weights× Labels

- Now sort all the tokens in ascending order of their feature value.
- In the example here, you can see immediately that large feature values tend to be associated with the label $y_i = 1$, and small feature values with $y_i = 0$, though there's a lot of variability.



True Reject, False Reject, True Accept, False Accept

- If $h_j(x_i) = 0$, we say that the classifier "rejects" the token.
- If $h_j(x_i) = 1$, we say the classifier "accepts" the token.
- If $h_j(x_i) = y_i$, we call this a "true accept" or "true reject."
- If $h_j(x_i) \neq y_i$, we call it a "false accept" or "false reject."

	$f_j(x_i) = 0$	$f_j(x_i) = 1$
$y_i = 0$	TR	FA
$y_i = 1$	FR	TA

True Reject, False Reject, True Accept, False Accept

- If $p_j = +1$, the classifier **accepts** any token with $f_j(x_i) < \theta_j$:

$$\Pr(FA|p_j = 1) = \Pr(f_j(x_i) < \theta_j, y_i = 0) = \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)(1 - y_i)$$

$$\Pr(TA|p_j = 1) = \Pr(f_j(x_i) < \theta_j, y_i = 1) = \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)y_i$$

- If $p_j = -1$, the classifier **rejects** any token with $f_j(x_i) < \theta_j$:

$$\Pr(TR|p_j = -1) = \Pr(f_j(x_i) < \theta_j, y_i = 0) = \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)(1 - y_i)$$

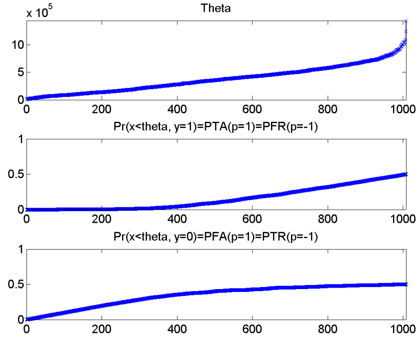
$$\Pr(FR|p_j = -1) = \Pr(f_j(x_i) < \theta_j, y_i = 1) = \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)y_i$$

True Reject, False Reject, True Accept, False Accept

The optimum values of p_j and θ_j are somehow related to these two curves:

$$\Pr(FA|p_j = 1) = \Pr(TR|p_j = -1) = \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)(1 - y_i)$$

$$\Pr(TA|p_j = 1) = \Pr(FR|p_j = -1) = \sum_{i:f_j(x_i)<\theta_j} w_j(x_i) y_i$$



Training Error

The probability of error is the probability of false accept, plus the probability of false reject. If $p_j = +1$:

$$
\begin{aligned}
\Pr(\text{Error}|p_j = +1) &= \Pr(FA|p_j = 1) + \Pr(FR|p_j = 1) \\
&= \Pr(FA|p_j = 1) + (\Pr(y_i = 1) - \Pr(TA|p_j = +1)) \\
&= \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)(1 - y_i) + P_1 - \sum_{i:f_j(x_i)<\theta_j} w_j(x_i) y_i \\
&= P_1 - \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)(2y_i - 1),
\end{aligned}
$$

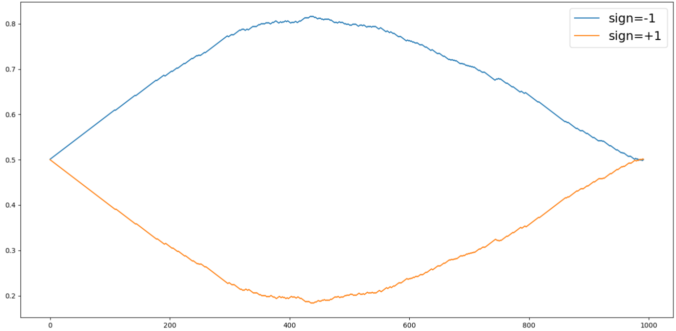
where $P_1 \equiv \Pr(y_i = 1)$. Similarly, if $p_j = -1$:

$$
\Pr(\text{Error}|p_j = -1) = P_0 + \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)(2y_i - 1)
$$

Minimum Training Error

$$\Pr(\text{Error}|p_j = 1, \theta_j) = P_1 - \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)(2y_i - 1)$$

$$\Pr(\text{Error}|p_j = -1, \theta_j) = P_0 + \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)(2y_i - 1)$$



Optimizing the Weak Classifier

So, given a particular feature $f_j = (\phi_{j,1}, \phi_{j,2}, \phi_{j,3}, \phi_{j,4}, o_{j,1}, o_{j,2})$, we can find the best weak classifier by calculating these two curves, and then choosing the value of p_j and θ_j that minimizes the error rate:

$$\Pr(\text{Error}|p_j = 1, \theta_j) = P_1 - \sum_{i: f_j(x_i) < \theta_j} w_j(x_i)(2y_i - 1)$$

$$\Pr(\text{Error}|p_j = -1, \theta_j) = P_0 + \sum_{i: f_j(x_i) < \theta_j} w_j(x_i)(2y_i - 1)$$

Outline

AdaBoost

- The AdaBoost algorithm ("adaptive boosting") is an algorithm that combines several weak classifiers in order to form a strong classifier.
- Suppose that $h_t(x) \in \{0,1\}$ is the t^{th} weak classifier
- Suppose that α_t is the confidence of the t^{th} weak classifier
- Then the strong classifier's decision is given by:

$$h(x) = \begin{cases} 1 & \sum_t \alpha_t h_t(x) > \frac{1}{2}\sum_t \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

AdaBoost

$$h(x) = \begin{cases} 1 & \sum_t \alpha_t h_t(x) > \frac{1}{2}\sum_t \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

- Notice that this is a kind of neural net. The first-layer excitation is $p_t\theta_t - p_t f_t(x)$, the first-layer nonlinearity is a unit step, and the second-layer weights are α_t.
- It's like a neural net during training time, but the training algorithm is different.

The AdaBoost Training Algorithm

The AdaBoost training algorithm is as follows:

1. **Initialize:** Assign all training tokens the same weight.
2. **Iterate:** for $t = 1, 2, \ldots$:
   1. Exhaustively test every feature f_j:
      - Find p_j, and θ_j to minimize the weighted training corpus error.
   2. Set h_t equal to the h_j that had the lowest error.
   3. Decrease the weight of the correctly classified tokens, and increase the weight of the incorrectly classified tokens.

The result is that each new classifier, h_t, is encouraged to try to fix the mistakes of all the classifiers that came before it.

The AdaBoost Training Algorithm

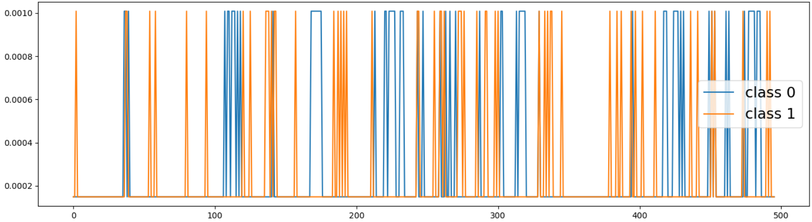
1. **Initialize:** $w_1(x_i) = 1, \quad 1 \leq i \leq n$.
2. **Iterate:** for $t = 1, 2, \ldots$
   1. Rescale the weights so they sum to one
   2. Exhaustively test every possible feature. Find f_t, p_t, and θ_t to minimize
      $$\epsilon_t = \sum_i w_t(x_i)|y_i - h_t(x_i)|$$
   3. If any training token was correctly classified, decrease its weight by
      $$w_{t+1}(x_i) = \beta_t w_t(x_i), \quad \beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$$

The AdaBoost Training Algorithm

```
for t in range(T):
  w = w / np.sum(w)
  for theta1 in [0,1/6,2/6,3/6,4/6,5/6]:
    ...
      for theta3 in [1/6,2/6,...,1-theta1]:
        ...
          for o2 in [1,...,4-o1]:
            epsilon = error(theta1,...)
            if epsilon < epsilon_best[t]:
              f[t] = (theta1, theta2, ..., o2)
  beta[t] = epsilon_best[t] / (1-epsilon_best[t])
  w[h == y] *= beta[t]
```

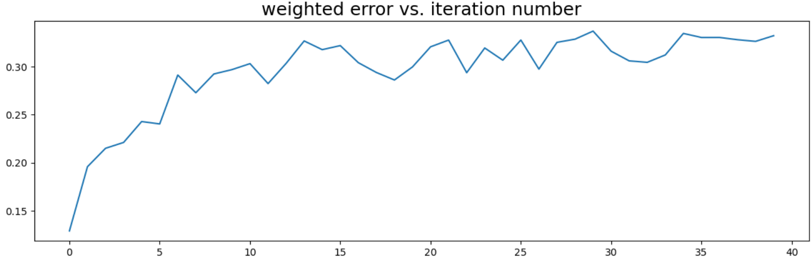
The AdaBoost Training Algorithm

Thus, for example, after the first iteration, the weights $w_2(x)$ have two different magnitudes: those that were correctly classified by $h_1(x)$, and those that were incorrectly classified:



The AdaBoost Training Algorithm

- The weighted error rate tends to be lowest in the first iteration, and get worse as t gets larger:



- That's because, as t increases, the tokens that are hard to classify get higher and higher weights, while the easy tokens count for less and less.

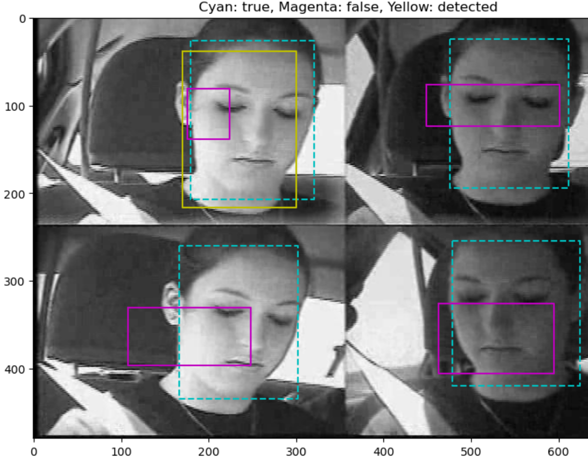
AdaBoost Testing

In order to test the trained AdaBoost classifier, we use

$$h(x) = \begin{cases} 1 & \sum_t \alpha_t h_t(x) > \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = -\ln \beta_t$.

Cyan = correct, Magenta = incorrect, Yellow = detected



Outline

Haar-like Features: Convolutions with Super-low Computation because of the Integral Image

$$II[m,n] = \sum_{m'=1}^{m} \sum_{n'=1^n} I[m',n'], \quad 1 \le m \le M, 1 \le n \le N$$

$$\sum_{m'=m}^{m+h} \sum_{n'=n}^{n+w} I[m',n'] = II[m+h,n+w] - II[m,n+w] - II[m+h,n] + II[m,n]$$

The Weak Classifier: Sort the Training Corpus by Feature Value, Choose the Threshold with Lowest Error

$$P_1 - \Pr(\text{Error}|p_j = 1, \theta_j) =$$
$$\Pr(\text{Error}|p_j = -1, \theta_j) - P_0 = \sum_{i:f_j(x_i)<\theta_j} w_j(x_i)(2y_i - 1)$$

AdaBoost: Each Weak Classifier Tries to Correct the Mistakes of the Ones that Came Before

for $t = 1, 2, \ldots$

1. Rescale the weights so they sum to one
2. Find f_t, p_t, and θ_t to minimize
$$\epsilon_t = \sum_i w_t(x_i)|y_i - h_t(x_i)|$$
3. If any training token was correctly classified, decrease its weight by
$$w_{t+1}(x_i) = \beta_t w_t(x_i), \quad \beta_t = \frac{\epsilon_t}{1-\epsilon_t}$$

The strong classifier is

$$h(x) = \begin{cases} 1 & \sum_t \alpha_t h_t(x) > \frac{1}{2}\sum_t \alpha_t \\ 0 & \text{otherwise} \end{cases}, \qquad \alpha_t = -\ln \beta_t$$