Adalines

Primitive Artificial Neurons
The Adaline (Adaptive Linear Neuron or Adaptive Linear Element) is a model similar to the Perceptron. There are several variations:

- One has the threshold function similar to a perceptron.
- Another uses a pure linear function with no threshold.
Adaline Inventors
Bernard Widrow and Marcian (Ted) Hoff

Bernard Widrow, Professor Emeritus of E.E., Stanford University

Marcian Hoff
Co-inventor of Patent 3,821,715
*Microprocessor Concept and Architecture*
With or without the threshold, the Adaline is trained based on the output of the linear function rather than the final output.
The catch here is that we have to state the **desired** value in terms of the output of the linear part, rather than the output after the threshold.

What is this for a classifier?
A reasonable approach is to use a *nominal* value such as -0.5 as desired for a “no” classification and a +0.5 for a “yes” classification.
The formula for Adaline weight updating is very similar to the Perceptron:

Add to the weights $\Delta w$ where

$\Delta w = [-1, x_1, x_2, \ldots, x_n]$

Adaline learning rule

only now the error is not limited to 1, -1, 0 as before; it can have a fractional value, since it is based on the output of the linear part of the device.
The weight update formula for the Adaline will be justified eventually.

One major difference from this vs. the Perceptron is that a learning rate of 1 won’t generally be acceptable. It will need to be smaller, say 0.01. There is a theory that tells us how large we can make it.
Adaline Example

We’ll use the same example as before. But now we’ll train on the output of the linear portion and target for +1 for a “yes” answer and -1 for a “no” answer.

- (4, 5) +1
- (6, 1) +1
- (4, 1) -1
- (1, 2) -1

Try a learning rate of 0.01.
# Adaline Training Example

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 1, -1)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 6, 1)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 4, 1)</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 1, 2)</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Adaline Training Example

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 1, -1)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td>-1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>(-1, 6, 1)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 4, 1)</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 1, 2)</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td>(0.0358, 0.727, -0.943)</td>
</tr>
</tbody>
</table>

We stopped (after 500 epochs) at

(0.67, 0.21, 0.17)

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0.67, 0.21, 0.17)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td>1.02</td>
<td>-0.02</td>
</tr>
<tr>
<td>(-1, 6, 1)</td>
<td>1</td>
<td></td>
<td>0.75</td>
<td>0.25</td>
</tr>
<tr>
<td>(-1, 4, 1)</td>
<td>-1</td>
<td></td>
<td>0.33</td>
<td>-1.33</td>
</tr>
<tr>
<td>(-1, 1, 2)</td>
<td>-1</td>
<td></td>
<td>-0.12</td>
<td>-0.88</td>
</tr>
</tbody>
</table>

With a threshold of 0.67, this would correctly separate the samples.
Adaline Convergence (2)

- The Adaline admits a more refined stopping criterion:
  The **Mean-Squared Error (MSE)** is the average of the squares of the error taken over all samples. Squaring makes the measure insensitive to the sign of the error. It also provides certain analytic properties.

- This quantity ideally converges toward a specific minimum (which might never be exactly attained). The algorithm can be set to stop when the MSE reaches a desired value.
Alternate Rule Names

Because the Adaline rule minimizes MSE, it is sometimes called the “LMS rule” [LMS = “least mean square”].

The term “Delta rule” is also sometimes used, although this will be seen to be a rule for a more general class of networks.
Minimizing Error

- Consider the task of finding the minimum of a function of one variable.

- One standard method for doing this, if the derivative of the function is known, is Newton’s method, which entails constructing a tangent line from a current estimate, then using the intersection of the line with the origin for the next.
Gradient Descent

- Gradient descent is another method for finding the minimum.

- It consists of computing the gradient of the function, then taking a small step in the direction of negative gradient, which hopefully corresponds to decreased function value, then repeating for the new value of the dependent variable.
Gradient Descent

MSE

Negative gradient direction

weight
Gradient Descent

- A single dimension for weights (including bias) is atypical.
- So the previous diagram is mainly to enhance our intuition.
- For the general case, the gradient is a vector of gradient components, one for each weight (including bias).
Error vs. 2-D Weight Space
(one weight is threshold or -bias)
2-D Gradient Descent
2-D Gradient Descent
Computing Gradients

- $\text{MSE} = J(w) = \frac{1}{n} \sum (\text{desired-actual})^2$ where $\sum$ is over $n$ samples.
- desired is a fixed constant for each sample.
- actual $= \sum w_jx_j$ (sum over input lines)
- So $J(w) = \frac{1}{n} \sum (\text{desired} - \sum w_jx_j)^2$
On-Line Approximation to Gradient

- On-line means based on a single sample, rather than batch, which means using all samples

\[ J \approx (d - \sum w_j x_j)^2 \]  
\[ (d = \text{desired}) \]

- \( i^{th} \) gradient component = \( \frac{\partial J}{\partial w_i} \)
  
  \[ = \frac{\partial}{\partial w_i} (d - \sum w_j x_j)^2 \]
  
  \[ = 2 (d - \sum w_j x_j) \frac{\partial}{\partial w_i} (d - \sum w_j x_j) = -2 \sum x_i \]

\[ = \text{error, } \sum \]

\[ -x_i \]
Computing Gradients

- $i^{th}$ gradient component = $-2 \times x_i$
- However we want to move in the direction of negative gradient, tempered by the learning rate $\eta$, so:

  Amount to add to weight is
  \[
  \Delta w_i = 2 \times \eta \times x_i
  \]
  which we recognize as the LMS (Adaline) rule (2 can be folded into $\eta$).
Vector Version of the Analysis of Gradient Descent for Adaline

• \( \text{MSE} = J(w) = \mathbb{E}[(d - w^T x)^2] \) \hspace{1cm} (\( \mathbb{E} \) = expectation or mean averaged over samples)

\[
= \mathbb{E}[d^2] - 2dw^T x + w^T x x^T w \\
= \mathbb{E}[d^2] - \mathbb{E}[2dw^T x] + \mathbb{E}[w^T x x^T w] \\
= \mathbb{E}[d^2] - 2w^T \mathbb{E}[dx] + w^T \mathbb{E}[x x^T] w \\
= c - 2w^T h + w^T R w, \text{ for appropriate const. } c, h, R
\]
Vector Analysis of Gradient Descent for Adaline

- $J(w) = c - 2w^T h + w^T R w$

where $c = E[d^2]$, $h = E[d x]$, $R = E[x x^T]$

- This is a **quadratic form** in $w$ with coefficients derived from the data vectors $x$.

- $R$ is called the (auto-)**correlation matrix**.
Standard Quadratic Form

- $J(w) = c + w^Tb + (1/2)w^TAw$

where

- $c = E[d^2]$
- $b = -2E[d\times]$ 
- $A = 2E[x\times^T]$

- $A$ is called the **Hessian** matrix. It is the matrix of 2nd partial derivatives of the surface.
Analytic Gradient

\[ \frac{\partial}{\partial w} J(w) = \frac{\partial}{\partial w} (c + w^T b + (1/2)w^T A w) = b + A w \]

It can be shown that if \( J \) has a \textbf{minimum}, it will be at a point \( w^* \) where \( \frac{\partial}{\partial w} J(w^*) = 0 \), i.e. \( b + A w^* = 0 \)

\[ \text{i.e. } w^* = A^{-1} b \]

where \( A = 2E[x x^T], \ b = -2E[dx] \)
In general, $w^* = A^{-1}b$ is a stable point. It may correspond to a minimum, maximum, or saddle.
Convergence of Gradient Descent for Adaline

- $\Delta w = 2 \cdot \mathbf{h} \cdot \mathbf{x}$

  $w(k+1) = w(k) + 2 \cdot \mathbf{h}(k) \cdot \mathbf{x}(k)$ \hspace{1cm} (at $k^{th}$ step)

- $E[w(k+1)] = E[w(k)] + 2 \cdot \mathbf{h} \cdot E[\mathbf{h}(k) \cdot \mathbf{x}(k)]$

  ... math ...

  $= (I - 2\cdot \mathbf{h}) \cdot E[w(k)] + 2\cdot \mathbf{h}$

- For convergence, the eigenvalues of the matrix $I - 2\cdot \mathbf{h}$ must be within the unit circle.
Convergence of Gradient Descent for Adaline

- If $\lambda_i$ is an eigenvalue of $R$, convergence requires $|1 - 2 \lambda \lambda_i| < 1$.
- which simplifies to $\lambda < 1/ \lambda_i$ for all eigenvalues $\lambda_i$, in particular for the maximum one.

$$\lambda < 1/ \lambda_{\text{max}}$$

Bound on learning rate for convergence of Adaline training by gradient descent

NASDAQ repeatedly uses $2/ \lambda_{\text{max}}$, possible because it doesn’t keep the factor of 2 in the learning rule.
Example
(from NND)

\[ \mathbf{x}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \quad \mathbf{d}_1 = \begin{bmatrix} -1 \end{bmatrix} \]

Sample 2
\[ \mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \quad \mathbf{d}_2 = \begin{bmatrix} 1 \end{bmatrix} \]

\[ \mathbf{R} = E[\mathbf{x}\mathbf{x}^T] = \frac{1}{2} \mathbf{x}_1 \mathbf{x}_1^T + \frac{1}{2} \mathbf{x}_2 \mathbf{x}_2^T \]

\[ \mathbf{R} = \frac{1}{2} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix} \]

\[ \mathfrak{l}_1 = 1.0, \quad \mathfrak{l}_2 = 0.0, \quad \mathfrak{l}_3 = 2.0 \]

\[ \mathfrak{l} < \frac{1}{\mathfrak{l}_{\text{max}}} = \frac{1}{2.0} = 0.5 \]
Training, 1st Epoch

Sample 1

\[ a(0) = W(0)p(0) = W(0)x_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix} = 0 \]

\[ \square(0) = d(0) - a(0) = d_1 - a(0) = -1 - 0 = -1 \]

\[ W(1) = W(0) + 2\square(0)x^T(0) \]

\[ W(1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.2 \end{bmatrix} + 2(0.2)(-1) \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}^T = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} \]
Sample 2  \( a^{(1)} = W^{(1)} p^{(1)} = W^{(1)} p_2 = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = -0.4 \)

\[
\begin{aligned}
\square^{(1)} &= d^{(1)} - a^{(1)} = d_2 - a^{(1)} = 1 - (-0.4) = 1.4
\end{aligned}
\]

\[
W^{(2)} = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} + 2(0.2)(1.4) \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}^T = \begin{bmatrix} 0.960.16 \end{bmatrix}
\]
3rd Epoch

\[ a^{(2)} = \mathbf{W}^{(2)} \mathbf{p}^{(2)} = \mathbf{W}^{(2)} \mathbf{p}_1 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0.64 \]

\[ \square^{(2)} = d^{(2)} - a^{(2)} = d_1 - a^{(2)} = -1 - (-0.64) = -0.36 \]

\[ \mathbf{W}^{(3)} = \mathbf{W}^{(2)} + 2 \square^{(2)} \mathbf{x}^T(2) = \begin{bmatrix} 1.1040 & 0.0160 & -0.0160 \end{bmatrix} \]

\[ \mathbf{W}(\cdot) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \]
“Learning Curve”

- **Small $\eta_1$**
  - Weight tracks showing a smooth decrease.

- **$\eta_2 > \eta_1$**
  - Weight tracks showing a faster decrease with a slight oscillation.

- **Too large $\eta_3 > \eta_2$**
  - Weight tracks showing oscillations and divergence.

---

**Graph**

- **$J_{\text{min}}$**
  - Minimum error as a function of number of iterations.

- **Increasing $\eta$**
  - Curve showing how error decreases with increasing learning rate.

---

**Legend**

- $w(0)$, $w(1)$, $w(k)$, $w^*$
  - Weight values at different iterations.

- $\eta$
  - Learning rate parameter.

- $J_{\text{min}}$
  - Minimum error value.
If learning rate too high, minimum solution won’t be achieved, even if there is no divergence.
Solutions to Rattling

- Rule of thumb: Use a learning rate 0.1 times the theoretical maximum, or

- Gradually decrease the learning rate, e.g. according to a pre-set schedule, or

- Adaptively set the learning rate:
  - If the MSE is taking big steps, make it smaller.
  - If the MSE is taking small steps, make it larger.
Generalizing the Adaline

- We have already discussed the problem of a threshold output in this picture
Suppose that we replace the threshold stage with a general function $f$ and revert to expressing desired in terms of its output:
Generalizing the Adaline

Consider again the derivation of the LMS rule:

\[ J(w) = \frac{1}{n} \sum (\text{desired} - f(\sum w_j x_j))^2 \]

\[ J \approx (d - f(\sum w_j x_j))^2 \quad (d = \text{desired}) \]

- \( i^{th} \) gradient component = \( \frac{\partial J}{\partial w_i} \)

\[
= \frac{\partial}{\partial w_i} (d - f(\sum w_j x_j))^2 \\
= 2 (d - f(\sum w_j x_j)) \frac{\partial}{\partial w_i} (d - f(\sum w_j x_j)) \\
= -2 \sum \frac{\partial}{\partial w_i} f(\sum w_j x_j) \\
= -2 \sum f'(\sum w_j x_j) \frac{\partial}{\partial w_i} \sum w_j x_j = -2 \sum x_i f'(\sum w_j x_j)
\]
Generalized LMS Rule
(or Delta Rule)

- $\Delta w = 2 e f'(\sum w_j x_j) x_i$
  assuming that $f$ has a derivative $f'$.

- $\sum w_j x_j$ is often called the “net” value or “activation” value, and $f$ the activation function.

- For the special case of $f$ being the identity function, this reduces to the LMS rule we had before.
Generalized LMS Rule
(or Delta Rule)

Why worry about this generalization?
\[ \Delta w = 2 e h f'(\sum w_j x_j) x_i \]

It will have a number important uses.
In the Adaline with threshold, we can’t very well treat the model analytically, due to the fact that we have a non-continuous function at the output.

But we can approximate the non-continuous function with a continuous one:
The “S” shape on the right of the previous slide is called a sigmoid curve.

This is a generic term and there are several different analytic functions that behave this way.
Logistic Sigmoid

- Logistic function ("logsig"—Matlab):
  \[ f(x) = \frac{1}{1+\exp(-ax)} \]

- \[ f'(x) = f(x)(1-f(x)) \]
Hyperbolic Sigmoid

- Hyperbolic tangent function ("tansig"): 
  \[ f(x) = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \]

- \( f'(x) = 1 - f^2(x) \)
Squashing Functions

- Sigmoids, step functions, and other functions that force their results to be in a limited range are called “squashing functions”.

- It is generally accepted that biological neural system is based on such functions, since there are physical limits to the response level.