% LMS: A stochastic gradient descent algorithm inspired by neurobiology

Abhipray Sahoo, Jake Kaplan, Stephane Remigereau

Stanford University 2019

**Motivation**

Dr. Bernard Widrow proposed a linear neuron model, Hebbian-LMS, that learns via % LMS.

- Inputs → firing rate of pre-synaptic neurons
- Weights → number of neuroreceptors at the dendrite of a living neuron
- Output → firing rate

Training → min \[ E = \sum (y_k - \theta^T x_k)^2 \]

Concentration of neuroreceptors in living neurons changes via synaptic scaling; weights of the neuron model \( \theta \) or \( J \) as a multiplicative factor instead of additively leading to the %LMS update rule. We explore properties, performance and extensions.

**Convergence**

Does it converge?

How does it do with different shapes of the quadratic cost function?

**Observation: Cost Function**

A neuron updates its weight \( \theta \) to minimize:

\[
\min \{ d(\theta_{k+1}, \theta_k) + \alpha J(\theta_{k+1}) \} \quad (1)
\]

Set the gradient w.r.t \( \theta_{k+1} \) to 0 after assuming

\[
\nabla \theta_{k+1} J(\theta_{k+1}) \sim \nabla \theta_k J(\theta_k)
\]

\[
\nabla \theta_k, d(\theta_{k+1}, \theta_k) + \alpha \nabla \theta_k J(\theta_k) = 0 \quad (2)
\]

\[
J(\theta_k) = \text{MSE} = ||y_k - \theta^T x_k||_2^2 \quad (3)
\]

%LMS → \[
\frac{d(\theta_{k+1}, \theta_k)}{d(\theta_k, \theta_k)} = ||\theta_{k+1} - \theta_k||_2^2 \quad \%
\]

%LMS updates to minimize relative change in weights. Big weights adapt faster.

**Observation: Non-negativity**

Weights have to be non-negative because

- Negative weights grow more and more negative.
- Zero weights stop changing due to multiplication.

To prevent the weight becoming negative, the learning rate is bounded:

\[
\alpha_k \leq \frac{1}{\epsilon_k x_k} \quad \forall k \quad (4)
\]

**Generalized Algorithm and Variance**

Extend % LMS for negative weights:

\[
\theta_{k+1} = (1 + \alpha \epsilon x_k \text{sign}(\theta_k)) \circ \theta_k \quad (5)
\]

\[
\text{sign}(\theta_k) = \begin{cases} 
1 & \theta_k \geq 0 \\
-1 & \theta_k < 0
\end{cases} \quad (6)
\]

Add noise to prevent convergence to 0:

\[
\theta_{k+1} = (1 + \alpha \epsilon x_k \text{sign}(\theta_k)) + \tilde{g}(\theta_k) \circ \theta_k \quad (7)
\]

\[
\tilde{g}(\theta_k) = \begin{cases} 
\mathcal{N}(0, \epsilon^2) & -\epsilon^2 < \theta_k < \epsilon^2 \\
0 & \text{else}
\end{cases} \quad (8)
\]

- Might not converge to small \( \theta^* \) if \( \epsilon \) too large.
- False convergence to 0 if \( \epsilon \) too small.

**Solution:** Set \( \epsilon_k = \sqrt{\alpha \nabla J(\theta_k)} = ||\alpha \epsilon x_k||_2 \).

**Results:** All % LMS models showed convergence improvement; \( \epsilon^2 = \nabla J \) minimized loss deviation.

**Applications of General % LMS**

<table>
<thead>
<tr>
<th>Logistic Regression Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialization</td>
</tr>
<tr>
<td>acceptance accuracy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Neural Network MNIST Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture: single hidden layer with 150 sigmoid units, softmax output layer</td>
</tr>
<tr>
<td>Training: epochs=20, batch size=20, learning rate=0.1, cross-entropy loss, 50K MNIST examples</td>
</tr>
<tr>
<td>Testing: 10K MNIST examples</td>
</tr>
</tbody>
</table>

**Acknowledgements**


**Next Steps**

- Derive convergence properties and learning curve analytically.
- Compare the performance of % LMS when classifying different distributions (so far only Gaussian and Poisson distributions analyzed).