Session #11

CMSC 409: Artificial Intelligence

http://www.people.vcu.edu/~mmanic/

Virginia Commonwealth University,
Fall 2023,
Dr. Milos Manic
(mmanic@vcu. edu)

1

CMSC 409: Artificial Intelligence

Session # 11

Topics for today

- Announcements
- Previous session review
- Least Mean Squares (LMS) learning
  - *LMS learning, derivation*
  - *Incremental vs. batch learning*
- Learning - the effect of transfer function

2

CMSC 409: *Artificial Intelligence*

Announcements — Session # 11

- Canvas
  - *New slides posted*
- Office hours zoom
  - *Zoom disconnects me after 45 mins of inactivity. Feel free to chat me via zoom if that happens and I will reconnect (zoom chat welcome outside of office hours as well)!*
- Project #2
  - *Deadline Oct. 3 (noon)*
- Midterm exam
  - *Oct. 19 (Thu)*
- Paper (optional)
  - *The 2nd draft due Oct. 10 (noon)*
  - *Literature review and updated problem description (check out the class paper instructions for the 2nd draft)*
- Subject line and signature
  - *Please use [CMSC 409] Last_Name Question*

Project 2, quick reminder

		Predicted condition	
	Total population = P + N	Big cars = 1 Positive (PP)	Small cars = 0 Negative (PN)
Actual condition	Big cars = 1 Positive (P)	True positive (TP)	False negative (FN)
Actual condition	Small cars = 0 Negative (N)	False positive (FP)	True negative (TN)

Least Mean Squares (LMS) learning

- ***LMS learning, derivation***
- *Incremental vs. batch learning*

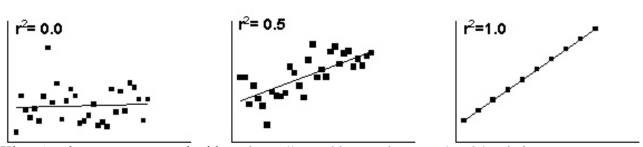
LMS (Least Mean Squares) Learning

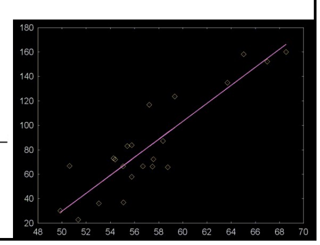
Regression analysis

- *We talked about the linear regression recently*
- *Minimizing of squared distances to each of the points*
- *We talked about Rosenblatt's Perceptron '58*
- *We talked about R^2 as measure of goodness-of-fit of regression*

LMS learning – more powerful than (original) perceptron:

- *Perceptron (original, with hard act. fun.) - sensitive to noise, often times decision boundaries lie close to the patterns.*
- *LMS minimizes mean square error, moves separation lines or (hype)planes as far from training patterns as possible!*





Khan Academy, easy to watch videos: https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/residuals-least-squares-rsquared/v/regression-line-example

LMS Learning (1)

Errors:

$$Err_1 = [d_1 - o_1]^2$$
$$Err_2 = [d_2 - o_2]^2$$
$$\cdots$$
$$Err_{np} = [d_{np} - o_{np}]^2$$

np – number of patterns
d – desired output
o – actual output

Need to combine all the errors in one error - **Total Error**:

$$TE = \sum_{p=1}^{np} [d_p - o_p]^2$$

total error for all patterns.

LMS learning (2)

Total Error:

$$TE = \sum_{p=1}^{np} [d_p - o_p]^2$$

where:

$$o_p = f(net_p) = f(w_1 x_1 + w_2 x_2 + \cdots + w_{ni} x_{ni})$$

$$o_p = f(net_p) = f\left(\sum_{i=1}^{ni} w_i x_i\right)$$

$$p = 1, 2, \ldots, np.$$

np – number of patterns
ni – number of inputs
p – pattern number
f – activation (threshold) function

net_p – net for pattern *p*
o_p – output for pattern *p*
x_i – pattern for input *i*
w_i – weight for input *i*

LMS learning (3)

The task is to find proper values of weight set (w_1 w_2 ... w_{ni}), in order to minimize TE!

$$TE = \sum_{p=1}^{np} \left[d_p - o_p\right]^2 \qquad o_p = f\left(net_p\right) = f\left(\sum_{i=1}^{ni} w_i x_i\right)$$

$$o_p = f\left(w_1 x_1 + w_2 x_2 + \cdots + w_{ni} x_{ni}\right)$$

Minimize total error by finding gradient of *TE* along w_i:

$$\frac{d(TE)}{dw_i} = -\sum_{p=1}^{np} 2\left[d_p - o_p\right] \frac{do_p}{dw_i}$$

9

LMS learning (4)

$$TE = \sum_{p=1}^{np} \left[d_p - o_p\right]^2$$

Minimize total error by finding the gradient of TE along w_i:

$$\frac{d(TE)}{dw_i} = -2\sum_{p=1}^{np} \left[d_p - o_p\right] \frac{do_p}{dw_i}$$

(derivative along w_i*)*

where: $o_p = f(net_p(w_i))$

$$\left(\text{since } \frac{dnet_p}{dw_i} = x_{ip}, \frac{do_p}{dnet_p} = f'\right)$$

$$\frac{do_p}{dw_i} = \frac{do_p}{dnet_p} \frac{dnet_p}{dw_i} = f' x_{ip}$$

f ` – slope of activation function

np – number of patterns

ni – number of inputs

p – pattern number

x_{ip} – pattern for input *i and pattern p*

net_p – net for pattern *p*

o_p – output for pattern *p*

w_i – weight for input *i*

10

LMS learning (5)

Knowing: $\frac{d(TE)}{dw_i} = -\sum_{p=1}^{np} 2\left[d_p - o_p\right]\frac{do_p}{dw_i}$ $\qquad TE = \sum_{p=1}^{np}\left[d_p - o_p\right]^2$

and $\frac{do_p}{dw_i} = \frac{do_p}{dnet_p}\frac{dnet_p}{dw_i} = f' x_{ip}$

derivative of *TE* becomes: $$\frac{d(TE)}{dw_i} = -\sum_{p=1}^{np} 2\left[d_p - o_p\right] f' x_{ip}$$

In **LMS rule** *(Widrow-Hoff 1962)*: $TE = \sum_{p=1}^{np}\left[d_p - net_p\right]^2$

(net instead of actual output) $$\frac{d(TE)}{dw_i} = -\sum_{p=1}^{np} 2\left[d_p - o_p\right] x_{ip}$$ since ($f'=1$):

At that time hard threshold transfer function used => f'=0 & grad=0 !

LMS learning rule (6)

$$gradient \Rightarrow \frac{d(TE)}{dw_i} = -\sum_{p=1}^{np} 2\cdot\left[d_p - o_p\right]\cdot x_{ip}$$

x_{ip} – pattern for input *i and pattern p*

It order to minimize total error *TE* we have to modify parameters against their gradient:

$$\Delta w_i = -\alpha \cdot gradient$$

where $\boldsymbol{\alpha}$ defines how far we have to go along the gradient in a given direction:

$$\Delta w_i = -\alpha \cdot \left(-\sum_{p=1}^{np} 2\cdot\left[d_p - o_p\right]\cdot x_{ip}\right)$$

We are looking for the steepest descent downhill along the gradient!

LMS learning rule (8)

Knowing incremental weight change for LMS rule:

$$\Delta w_{ip} = 2 \cdot \alpha \cdot [d_p - o_p] \cdot x_{ip}$$

(corresponding change for only one pattern)

….we realize the similarity with the **perceptron learning rule**:

$$\Delta \mathbf{w}_i = \alpha\, \mathbf{x}_i \left(d - o\right)$$

(perceptron learning rule)

or: $\Delta \mathbf{w}_i = \alpha\, \mathbf{x}_i \left(d - \mathrm{sign}(net)\right)$.

(hard threshold function)

LMS equation for batch training :

(accumulating changes - do not change weights after every pattern)

$$\Delta w_i = 2 \cdot \alpha \sum_{p=1}^{np} \left[d_p - o_p\right] x_{ip}$$

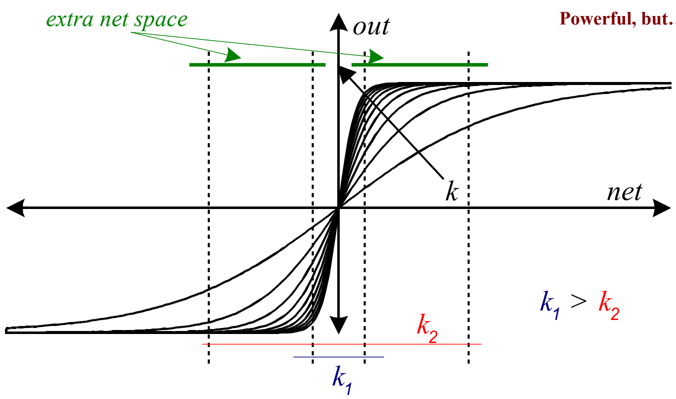
LMS batch training

Assuming f'=1, perceptron rule is essentially the same as the LMS rule…

Learning - the effect of transfer function

Bipolar soft activation function, net vs. out

Powerful, but…vanishing gradient!



$$f_{bip}(k \cdot net) = tanh(k \cdot net) = \frac{2}{1+exp(-k \cdot net)} - 1$$

k grows larger => soft becomes hard activation function

k gets smaller => soft becomes linear activation function, y axis

Unipolar soft activation function, derivation

***Unipolar soft activation function*:**

$$\boxed{o_{uni} = f_{uni}(k \cdot net) =}\left(\tanh(k \cdot net)+1\right)/2 = \ldots$$

…which can be in couple of steps reduced to…

$$\ldots = \frac{exp(k \cdot net)}{exp(k \cdot net) + exp(-k \cdot net)} = \frac{1}{1+exp(-2 \cdot k \cdot net)} =>$$

Finally:
$$\boxed{o_{uni} = f_{uni}(k \cdot net) = \frac{1}{1+exp(-k \cdot net)}}$$

First derivative:

$$\boxed{f'_{uni}(k \cdot net) = \left(\left(1+exp(-k \cdot net)\right)^{-1}\right)' = \ldots}$$

…that after couple of steps…

$$\ldots = k \cdot \frac{exp(-k \cdot net)+1-1}{\left(1+exp(-k \cdot net)\right)^2} = k \cdot \frac{1}{1+exp(-k \cdot net)} \cdot \frac{exp(-k \cdot net)+1-1}{1+exp(-k \cdot net)} =>$$

Finally:
$$\boxed{f'_{uni}(k \cdot net) = k \cdot o \cdot (1-o)}$$

Bipolar soft activation function, derivation

Bipolar soft activation function:

$$\boxed{f_{bip}(k\cdot net)=tanh(k\cdot net)}=\frac{exp(k\cdot net)-exp(-k\cdot net)}{exp(k\cdot net)+exp(-k\cdot net)}=$$

$$=\frac{1-exp(-2\cdot k\cdot net)}{1+exp(-2\cdot k\cdot net)}=\ldots$$

$$\boxed{\ldots=\frac{2}{1+exp(-2\cdot k\cdot net)}-1}$$

which can be reduced through couple of manipulations to

$$\left(\begin{array}{l}\frac{1-exp(-2\cdot k\cdot net)}{1+exp(-2\cdot k\cdot net)}=/_{exp(-2\cdot k\cdot net)=Y}/=\frac{1-Y+(1+Y-1-Y)}{1+Y}=\\ =\frac{2+(-1-Y)}{1+Y}=\frac{2}{1+Y}-1=\frac{2}{1+exp(-2\cdot k\cdot net)}-1\end{array}\right)$$

First derivative:

$$\boxed{f'_{bip}(k\cdot net)=\left[tanh(k\cdot net)\right]'}=\frac{4\cdot k\cdot exp(-2\cdot k\cdot net)}{(1+exp(-2\cdot k\cdot net))^2}=\ldots$$

k in denominator grows (²)

while in numerator linearly:

(k larger => f' smaller)

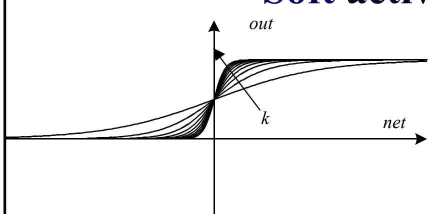
...that after couple of steps... $\ldots=k\cdot\left(1-(tanh(k\cdot net))^2\right)$

Finally:

$$\boxed{f'_{bip}(k\cdot net)=k\cdot\left(1-f^2_{bip}(k\cdot net)\right)}$$

17

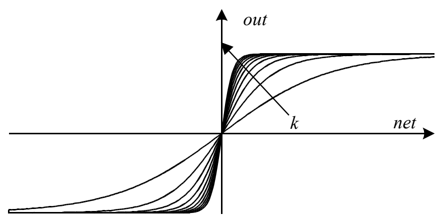
Soft activation functions



a)

- *S-curve, mapping net to values between (0, 1)*
- *0.5 centered*
- *Vanishing gradient!*

$$\boxed{o_{uni}=f_{uni}(k\cdot net)=\frac{1}{1+exp(-k\cdot net)}}$$



b)

- *S-curve, mapping net to values between (-1, 1)*
- *tanh is zero-centered, capturing positive and negative net/out correlations*
- *Vanishing gradient!*

$$\boxed{\begin{array}{l}o_{bip}=f_{bip}(k\cdot net)=tanh(k\cdot net)=\\ =\frac{2}{1+exp(-2\cdot k\cdot net)}-1\end{array}}$$

$$\boxed{f'_{uni}(k\cdot net)=k\cdot o\cdot(1-o)}\quad\boxed{f'_{bip}(k\cdot net)=k\cdot\left(1-o^2_{bip}\right)}$$

Unipolar

(or sigmoid w/ gain incorporated, 0 to 1)

Bipolar

(or hyperbolic tangent w/ gain incorp., 1- to +1)

18

Things to remember…

- **Soft act. function**
  - *Derivation: start with bipolar soft act. fun (definition of tanh); Do it yourself to become familiar with derivation*
  - *Neuron (or network) being in saturation boils down to output(s) being insensitive to change of inputs; To bring back network into a "learning" mode, you can reduce gain or scale down weights.*
  - *Decreasing gain may "restart" learning of a network. Too small gain will however lead to slow learning (better trade-off when network in "flat-spot".*
- **Vanishing gradient**
  - *Gradient (first derivative of soft activation function) plays a role in many NN learning rules (weight increment directly proportional to first derivative).*
  - *Gradient becomes very small (close to zero) when network goes into saturation. This is another consequence of soft activation function.*
  - *Decreasing gain will effectively increase first derivative, hence "restart" the learning ability of a network.*