## 15-494/694: Cognitive Robotics

## Dave Touretzky

Lecture 12:
Backpropagation Learning


Image from http://www.futuristgerd.com/2015/09/10

## Training A Linear Unit

$$
y=w_{0}+w_{1} \cdot x
$$




## LMS / Widrow-Hoff Rule



$$
\begin{gathered}
\Delta w_{i}=-\eta(y-d) x_{i} \\
\eta \text { is a learning rate } \\
(\text { could use } \eta=0.1)
\end{gathered}
$$

Works fine for a single layer of trainable weights. What about multi-layer networks?

## With Linear Units, Multiple Layers Don't Add Anything

$\bar{y}$
$\uparrow \boldsymbol{U}: \quad 2 \times 3$ matrix


$$
\bar{y}=\boldsymbol{U} \times(\boldsymbol{V} \bar{x})=\underbrace{(\boldsymbol{U} \times \boldsymbol{V})}_{2 \times 4} \overline{\bar{x}}
$$

$$
\uparrow \boldsymbol{V}: \quad 3 \times 4 \text { matrix }
$$



Linear operators are closed under composition. Equivalent to a single layer of weights $\boldsymbol{W}=\boldsymbol{U} \times \boldsymbol{V}$

But with non-linear units, extra layers add computational power.

## What Can be Done with Non-Linear (e.g., Threshold) Units?

$$
y=h\left(w_{0}+w_{1} \cdot x_{1}+w_{2} \cdot x_{2}\right)
$$

1 layer of trainable weights


separating hyperplane

2 layers of trainable weights


convex polygon region

3 layers of trainable weights


## How Do We Train A Multi-Layer Network?



Can't use perceptron training algorithm because we don't know the 'correct' outputs for hidden units.

## How Do We Train A Multi-Layer Network?

Define sum-squared error:

$$
E=\frac{1}{2} \sum_{p}\left(d^{p}-y^{p}\right)^{2}
$$

Use gradient descent error minimization:


$$
\Delta w_{i j}=-\eta \frac{\partial E}{\partial w_{i j}}
$$

Works if the nonlinear transfer function is differentiable.

## Deriving the LMS or "Delta" Rule As Gradient Descent Learning

$$
\begin{gathered}
y=\sum_{i} w_{i} x_{i} \\
E=\frac{1}{2} \sum_{p}\left(d^{p}-y^{p}\right)^{2} \\
y
\end{gathered}
$$

$$
\frac{d E}{d y}=y-d
$$

$$
\frac{\partial E}{\partial w_{i}}=\frac{d E}{d y} \cdot \frac{\partial y}{\partial w_{i}}=(y-d) x_{i}
$$

$$
\Delta w_{i}=-\eta \frac{\partial E}{\partial w_{i}}=-\eta(y-d) x_{i}
$$

How do we extend this to two layers?

## Switch to Smooth Nonlinear Units

$$
\begin{aligned}
\text { net }_{j} & =\sum_{i} w_{i j} y_{i} \\
y_{j} & =\mathrm{g}\left(\text { net }_{j}\right) \quad g \text { must be differentiable }
\end{aligned}
$$

Common choices for $g$ :

$$
\begin{aligned}
& g(x)=\frac{1}{1+e^{-x}} \\
& g^{\prime}(x)=g(x) \cdot(1-g(x))
\end{aligned}
$$

$$
g(x)=\tanh (x)
$$

$$
g^{\prime}(x)=1 / \cosh ^{2}(x)
$$



## Gradient Descent with Nonlinear Units



$$
y=g(n e t)=\tanh \left(\sum_{i} w_{i} x_{i}\right)
$$

$$
\frac{d E}{d y}=(y-d), \quad \frac{d y}{d n e t}=1 / \cosh ^{2}(n e t), \quad \frac{\partial n e t}{\partial w_{i}}=x_{i}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial w_{i}} & =\frac{d E}{d y} \cdot \frac{d y}{d n e t} \cdot \frac{\partial n e t}{\partial w_{i}} \\
& =(y-d) / \cosh ^{2}\left(\sum_{i} w_{i} x_{i}\right) \cdot x_{i}
\end{aligned}
$$

## Now We Can Use The Chain Rule



$$
\begin{gathered}
\frac{\partial E}{\partial y_{k}}=\left(y_{k}-d_{k}\right) \\
\delta_{k}=\frac{\partial E}{\partial n e t_{k}}=\left(y_{k}-d_{k}\right) \cdot g^{\prime}\left(n e t_{k}\right) \\
\frac{\partial E}{\partial w_{j k}}=\frac{\partial E}{\partial n e t_{k}} \cdot \frac{\partial n e t_{k}}{\partial w_{j k}}=\frac{\partial E}{\partial n e t_{k}} \cdot y_{j} \\
\frac{\partial E}{\partial y_{j}}=\sum_{k}\left(\frac{\partial E}{\partial n e t_{k}} \cdot \frac{\partial n e t_{k}}{\partial y_{j}}\right) \\
\delta_{j}=\frac{\partial E}{\partial n e t_{j}}=\frac{\partial E}{\partial y_{j}} \cdot g^{\prime}\left(\text { net }_{j}\right) \\
\frac{\partial E}{\partial w_{i j}}=\frac{\partial E}{\partial n e t_{j}} \cdot y_{i}
\end{gathered}
$$

## Weight Updates

$$
\begin{gathered}
\frac{\partial E}{\partial w_{j k}}=\frac{\partial E}{\partial n e t_{k}} \cdot \frac{\partial n e t_{k}}{\partial w_{j k}}=\delta_{k} \cdot y_{j} \\
\frac{\partial E}{\partial w_{i j}}=\frac{\partial E}{\partial n e t_{j}} \cdot \frac{\partial n e t_{j}}{\partial w_{i j}}=\delta_{j} \cdot y_{i} \\
\Delta w_{j k}=-n \cdot \frac{\partial E}{\partial w_{j k}} \quad \Delta w_{i j}=-n \cdot \frac{\partial E}{\partial w_{i j}}
\end{gathered}
$$

## Function Approximation


$3 n+1$ free parameters for $n$ hidden units

## Encoder Problem



## 5-2-5 Encoder Problem



Training patterns:

| $A:$ | 0 | 0 | 0 | 0 | 1 |
| :--- | :--- | :--- | :--- | :--- | :--- |
| $B:$ | 0 | 0 | 0 | 1 | 0 |
| $C:$ | 0 | 0 | 1 | 0 | 0 |
| $D:$ | 0 | 1 | 0 | 0 | 0 |
| $E:$ | 1 | 0 | 0 | 0 | 0 |

Hidden code: 0,2
2,0
1,-1
-1,1
$-1,0$


## Flat Spots

If weights become large, net ${ }_{j}$ becomes large, derivative of $g()$ goes to zero.

## flat spot



Fahlman's trick: add a small constant to $g^{\prime}(x)$ to keep the derivative from going to zero. Typical value is 0.1 .

## Momentum

Learning is slow if the learning rate is set too low.
Gradient may be steep in some directions but shallow in others.

Solution: add a momentum term $\alpha$.

$$
\Delta w_{i j}(t)=-\eta \frac{\partial E}{\partial w_{i j}(t)}+\alpha \cdot \Delta w_{i j}(t-1)
$$

Typical value for $\alpha$ is 0.5 .
If the direction of the gradient remains constant, the algorithm will take increasingly large steps.

## Momentum Illustration

Hertz, Krogh \& Palmer figs. 5.10 and 6.3: gradient descent on a quadratic error surface $E$ (no neural net) involved:

$$
\begin{aligned}
& E=x^{2}+20 y^{2} \\
& \frac{\partial E}{\partial x}=2 \mathrm{x}, \quad \frac{\partial E}{\partial y}=40 \mathrm{y} \\
& \text { Initial }[x, y]=[-1,1] \text { or }[1,1]
\end{aligned}
$$



MNIST Dataset
-60,000 labeled handwritten digits

- 28 x 28 pixel grayscale images

$$
\begin{array}{llllllllllllllll}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\
8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9
\end{array}
$$

## Recognition With a Linear Network



## Learned Weights to Output Units



Training set performance: $89 \%$ correct.

## TensorFlow Playground

Google's interactive backprop simulator. https://playground.tensorflow.org
DATA
Which dataset do
you want to use?
Noise: 0
Ratio of training to
nest data: $50 \%$

OUTPUT
Test loss 0.508
Training loss 0.508


