Recurrent Neural Nets

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing, Fall 2020

Outline

Basics of DSP: Filtering

$$y[n] = \sum_{m=-\infty}^{\infty} h[m]x[n-m]$$

$$Y(z) = H(z)X(z)$$

Finite Impulse Response (FIR)

$$y[n] = \sum_{m=0}^{N-1} h[m]x[n-m]$$

The coefficients, $h[m]$, are chosen in order to optimally position the $N-1$ zeros of the transfer function, r_k, defined according to:

$$H(z) = \sum_{m=0}^{N-1} h[m]z^{-m} = h[0]\prod_{k=1}^{N-1}\left(1 - r_k z^{-1}\right)$$

Infinite Impulse Response (IIR)

$$y[n] = \sum_{m=0}^{N-1} b_m x[n-m] + \sum_{m=1}^{M-1} a_m y[n-m]$$

The coefficients, b_m and a_m, are chosen in order to optimally position the $N-1$ zeros and $M-1$ poles of the transfer function, r_k and p_k, defined according to:

$$H(z) = \frac{\sum_{m=0}^{N-1} b_m z^{-m}}{1 - \sum_{m=1}^{M-1} a_m z^{-m}} = b_0 \frac{\prod_{k=1}^{N-1}\left(1 - r_k z^{-1}\right)}{\prod_{k=1}^{M-1}\left(1 - p_k z^{-1}\right)}$$

STABILITY: If any of the poles are on or outside the unit circle ($|p_k| \geq 1$), then $y[n] \rightarrow \infty$, even with finite $x[n]$.

Outline

Convolutional Neural Net = Nonlinear(FIR)

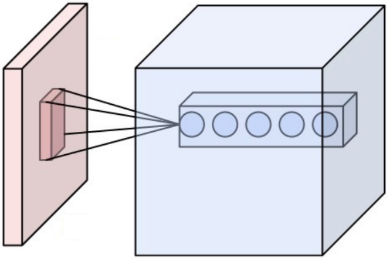


Image CC-SA-4.0 by Aphex34, `https://commons.wikimedia.org/wiki/File:Conv_layer.png`

Convolutional Neural Net = Nonlinear(FIR)

$$\hat{y}[n] = g\left(\sum_{m=0}^{N-1} w[m]x[n-m]\right)$$

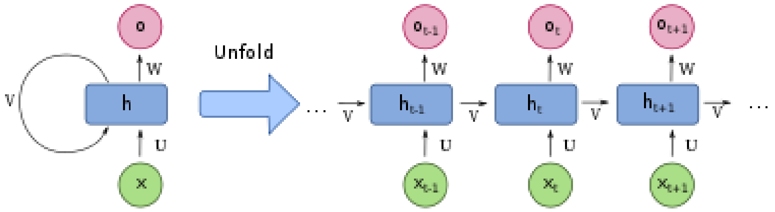
The coefficients, $w[m]$, are chosen to minimize some kind of error. For example, suppose that the goal is to make $\hat{y}[n]$ resemble a target signal $y[n]$; then we might use

$$E = \frac{1}{2}\sum_{n=0}^{N}\left(\hat{y}[n] - y[n]\right)^2$$

and choose

$$w[n] \leftarrow w[n] - \eta\frac{dE}{dw[n]}$$

Recurrent Neural Net (RNN) = Nonlinear(IIR)



Recurrent Neural Net (RNN) = Nonlinear(IIR)

$$\hat{y}[n] = g\left(x[n] + \sum_{m=1}^{M-1} w[m]y[n-m]\right)$$

The coefficients, $w[m]$, are chosen to minimize the error. For example, suppose that the goal is to make $\hat{y}[n]$ resemble a target signal $y[n]$; then we might use

$$E = \frac{1}{2}\sum_{n=0}^{N}\left(\hat{y}[n] - y[n]\right)^2$$

and choose

$$w[m] \leftarrow w[m] - \eta\frac{dE}{dw[m]}$$

Outline

Review: Excitation and Activation

- The **activation** of a hidden node is the output of the nonlinearity (for this reason, the nonlinearity is sometimes called the **activation function**). For example, in a fully-connected network with outputs $\hat{y}_l$, weights $\vec{w}$, bias b, nonlinearity $g()$, and hidden node activations $\vec{h}$, the activation of the l^{th} output node is

$$\hat{y}_l = g\left(b_l + \sum_{k=1}^{p} w_{lk} h_k\right)$$

- The **excitation** of a hidden node is the input of the nonlinearity. For example, the excitation of the node above is

$$e_l = b_l + \sum_{k=1}^{p} w_{lk} h_k$$

Backprop = Derivative w.r.t. Excitation

- The **excitation** of a hidden node is the input of the nonlinearity. For example, the excitation of the node above is

$$e_l = b_l + \sum_{k=1}^{p} w_{lk} h_k$$

- The gradient of the error w.r.t. the weight is

$$\frac{dE}{dw_{lk}} = \epsilon_l h_k$$

where ϵ_l is the derivative of the error w.r.t. the l^{th} **excitation**:

$$\epsilon_l = \frac{dE}{de_l}$$

Backprop for Fully-Connected Network

Suppose we have a fully-connected network, with inputs $\vec{x}$, weight matrices $W^{(1)}$ and $W^{(2)}$, nonlinearities $g()$ and $h()$, and output $\hat{y}$:

$$e_k^{(1)} = b_k^{(1)} + \sum_j w_{kj}^{(1)} x_j, \quad h_k = g\left(e_k^{(1)}\right)$$

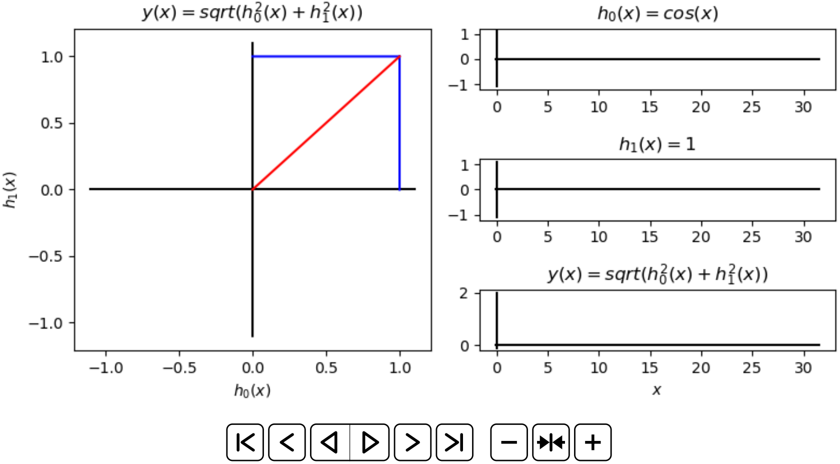
$$e_l^{(2)} = b_l^{(2)} + \sum_k w_{lk}^{(2)} h_k, \quad \hat{y}_l = h\left(e_l^{(2)}\right)$$

Then the back-prop gradients are the derivatives of E with respect to the **excitations** at each node:

$$\frac{dE}{dw_{lk}^{(2)}} = \epsilon_l h_k, \quad \epsilon_l = \frac{dE}{de_l^{(2)}}$$

$$\frac{dE}{dw_{kj}^{(1)}} = \delta_k x_j, \quad \delta_k = \frac{dE}{de_k^{(1)}}$$

Back-Prop Example



Back-Prop Example

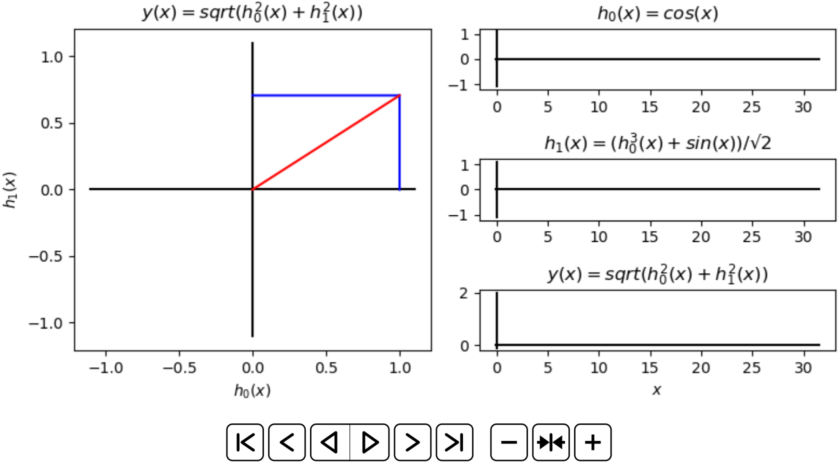
Suppose we have the following network:

$$h = \cos(x)$$
$$\hat{y} = \sqrt{1 + h^2}$$

Suppose we need $\frac{d\hat{y}}{dx}$. We find it as

$$\frac{d\hat{y}}{dx} = \frac{d\hat{y}}{dh}\frac{\partial h}{\partial x} = \left(\frac{h}{\sqrt{1+h^2}}\right)(-\sin(x))$$

Back-Prop Example



Back-Prop Example

Suppose we have the following network:

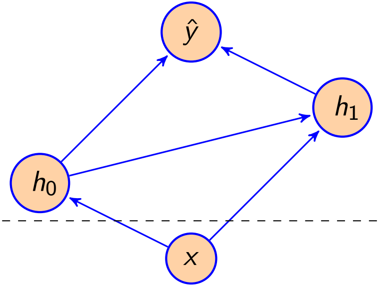
$$h_0 = \cos(x)$$

$$h_1 = \frac{1}{\sqrt{2}}\left(h_0^3 + \sin(x)\right)$$

$$\hat{y} = \sqrt{h_0^2 + h_1^2}$$

What is $\frac{d\hat{y}}{dx}$? How can we compute that?

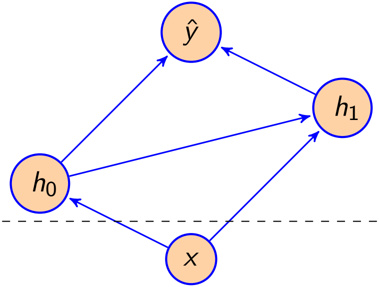
Causal Graphs for Neural Networks



We often show the causal graph for the chain rule using bubbles and arrows, as shown above. You can imagine the chain rule as taking a summation along any cut through the causal graph—for example, the dashed line shown above. You take the total derivative from $\hat{y}$ to the cut, and then the partial derivative from there back to x.

$$\frac{d\hat{y}}{dx} = \sum_{i=0}^{N-1} \frac{d\hat{y}}{dh_i} \frac{\partial h_i}{\partial x}$$

Causal Graphs for Neural Networks



$$\frac{d\hat{y}}{dx} = \sum_{i=0}^{N-1} \frac{d\hat{y}}{dh_i}\frac{\partial h_i}{\partial x}$$

For each h_i, we find the **total derivative** of $\hat{y}$ w.r.t. h_i, multiplied by the **partial derivative** of h_i w.r.t. x.

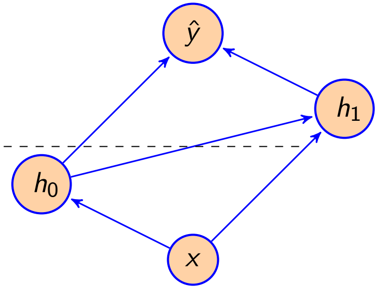
Back-Prop Example

First, we find $\frac{d\hat{y}}{dh_1}$:

$$\hat{y} = \sqrt{h_0^2 + h_1^2}$$

$$\frac{d\hat{y}}{dh_1} = \frac{h_1}{\sqrt{h_0^2 + h_1^2}}$$

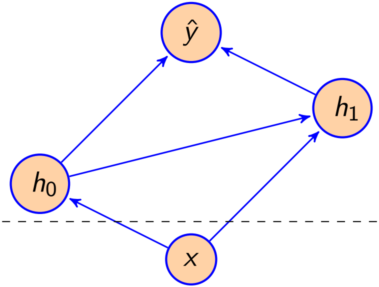
Back-Prop Example



Second, back-prop to find $\frac{d\hat{y}}{dh_0}$:

$$\frac{d\hat{y}}{dh_0} = \frac{\partial\hat{y}}{\partial h_0} + \frac{d\hat{y}}{dh_1}\frac{\partial h_1}{\partial h_0} = \frac{1}{\sqrt{h_0^2 + h_1^2}}\left(h_0 + \left(\frac{3}{\sqrt{2}}\right) h_0^2 h_1\right)$$

Back-Prop Example



Third, back-prop to find $\frac{d\hat{y}}{dx}$:

$$\frac{d\hat{y}}{dx} = \frac{d\hat{y}}{dh_1}\frac{\partial h_1}{\partial x} + \frac{d\hat{y}}{dh_0}\frac{\partial h_0}{\partial x}$$
$$= \left(\frac{h_1}{\sqrt{h_0^2 + h_1^2}}\right)\cos(x) - \left(\frac{\left(h_0 + \left(\frac{3}{\sqrt{2}}\right)h_0^2 h_1\right)}{\sqrt{h_0^2 + h_1^2}}\right)\sin(x)$$

Outline

Back-Prop in a CNN

Suppose we have a convolutional neural net, defined by

$$e[n] = \sum_{m=0}^{N-1} w[m]x[n-m]$$
$$\hat{y}[n] = g\left(e[n]\right)$$

then

$$\frac{dE}{dw[m]} = \sum_n \delta[n]x[n-m]$$

where $\delta[n]$ is the back-prop gradient, defined by

$$\delta[n] = \frac{dE}{de[n]}$$

Back-Prop in an RNN

Suppose we have a recurrent neural net, defined by

$$e[n] = x[n] + \sum_{m=1}^{M-1} w[m]\hat{y}[n-m]$$
$$\hat{y}[n] = g\left(e[n]\right)$$

then

$$\frac{dE}{dw[m]} = \sum_{n} \delta[n]\hat{y}[n-m]$$

where $\hat{y}[n-m]$ is calculated by forward-propagation, and then $\delta[n]$ is calculated by back-propagation as

$$\delta[n] = \frac{dE}{de[n]}$$

Outline

Partial vs. Full Derivatives

For example, suppose we want $\hat{y}[n]$ to be as close as possible to some target signal $y[n]$:

$$E = \frac{1}{2}\sum_n \left(\hat{y}[n] - y[n]\right)^2$$

Notice that E depends on $\hat{y}[n]$ in many different ways:

$$\frac{dE}{d\hat{y}[n]} = \frac{\partial E}{\partial \hat{y}[n]} + \frac{dE}{d\hat{y}[n+1]}\frac{\partial \hat{y}[n+1]}{\partial \hat{y}[n]} + \frac{dE}{d\hat{y}[n+2]}\frac{\partial \hat{y}[n+2]}{\partial \hat{y}[n]} + \ldots$$

Partial vs. Full Derivatives

In general,

$$\frac{dE}{d\hat{y}[n]} = \frac{\partial E}{\partial \hat{y}[n]} + \sum_{m=1}^{\infty} \frac{dE}{d\hat{y}[n+m]} \frac{\partial \hat{y}[n+m]}{\partial \hat{y}[n]}$$

where

- $\frac{dE}{d\hat{y}[n]}$ is the total derivative, and includes all of the different ways in which E depends on $\hat{y}[n]$.
- $\frac{\partial \hat{y}[n+m]}{\partial \hat{y}[n]}$ is the partial derivative, i.e., the change in $\hat{y}[n+m]$ per unit change in $\hat{y}[n]$ if all of the other variables (all other values of $\hat{y}[n+k]$) are held constant.

Partial vs. Full Derivatives

So for example, if

$$E = \frac{1}{2}\sum_n \left(\hat{y}[n] - y[n]\right)^2$$

then the partial derivative of E w.r.t. $\hat{y}[n]$ is

$$\frac{\partial E}{\partial \hat{y}[n]} = \hat{y}[n] - y[n]$$

and the total derivative of E w.r.t. $\hat{y}[n]$ is

$$\frac{dE}{d\hat{y}[n]} = \left(\hat{y}[n] - y[n]\right) + \sum_{m=1}^{\infty} \frac{dE}{d\hat{y}[n+m]} \frac{\partial \hat{y}[n+m]}{\partial \hat{y}[n]}$$

Partial vs. Full Derivatives

So for example, if

$$\hat{y}[n] = g(e[n]), \qquad e[n] = x[n] + \sum_{m=1}^{M-1} w[m]\hat{y}[n-m]$$

then the partial derivative of $\hat{y}[n+k]$ w.r.t. $\hat{y}[n]$ is

$$\frac{\partial \hat{y}[n+k]}{\partial \hat{y}[n]} = \dot{g}(e[n+k])w[k]$$

where we use the notation $\dot{g}(e) = \frac{dg}{de}$.

Synchronous Backprop vs. BPTT

The basic idea of back-prop-through-time is divide-and-conquer.

1. **Synchronous Backprop:** First, calculate the **partial derivative** of E w.r.t. the excitation $e[n]$ at time n, assuming that all other time steps are held constant.

$$\epsilon[n] = \frac{\partial E}{\partial e[n]}$$

2. **Back-prop through time:** Second, iterate backward through time to calculate the **total derivative**

$$\delta[n] = \frac{dE}{de[n]}$$

Synchronous Backprop in an RNN

Suppose we have a recurrent neural net, defined by

$$e[n] = x[n] + \sum_{m=1}^{M-1} w[m]\hat{y}[n-m]$$
$$\hat{y}[n] = g\left(e[n]\right)$$
$$E = \frac{1}{2}\sum_{n}\left(\hat{y}[n] - y[n]\right)^2$$

then

$$\epsilon[n] = \frac{\partial E}{\partial e[n]} = \left(\hat{y}[n] - y[n]\right)\dot{g}(e[n])$$

Back-Prop Through Time (BPTT)

Suppose we have a recurrent neural net, defined by

$$e[n] = x[n] + \sum_{m=1}^{M-1} w[m]\hat{y}[n-m]$$
$$\hat{y}[n] = g\left(e[n]\right)$$
$$E = \frac{1}{2}\sum_n \left(\hat{y}[n] - y[n]\right)^2$$

then

$$\delta[n] = \frac{dE}{de[n]}$$
$$= \frac{\partial E}{\partial e[n]} + \sum_{m=1}^{\infty} \frac{dE}{de[n+m]} \frac{\partial e[n+m]}{\partial e[n]}$$
$$= \epsilon[n] + \sum_{m=1}^{M-1} \delta[n+m] w[m] \dot{g}(e[n])$$

Outline

Conclusions

- Back-Prop, in general, is just the chain rule of calculus:

$$\frac{dE}{dw} = \sum_{i=0}^{N-1} \frac{dE}{dh_i} \frac{\partial h_i}{\partial w}$$

- Convolutional Neural Networks are the nonlinear version of an FIR filter. Coefficients are shared across time steps.
- Recurrent Neural Networks are the nonlinear version of an IIR filter. Coefficients are shared across time steps. Error is back-propagated from every output time step to every input time step.

$$\delta[n] = \frac{dE}{de[n]} = \frac{\partial E}{\partial e[n]} + \sum_{m=1}^{M-1} \delta[n+m] w[m] \dot{g}(e[n])$$

$$\frac{dE}{dw[m]} = \sum_n \delta[n] \hat{y}[n-m]$$