Partial and Total Derivatives

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 $y[n] \to \infty$, even with finite $x[n]$.

Outline

Convolutional Neural Net = Nonlinear(FIR)

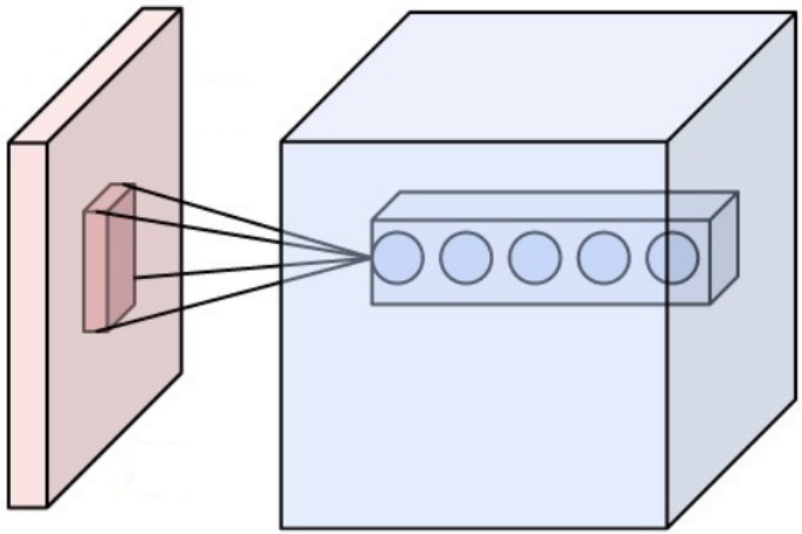


Image CC-SA-4.0 by Aphex34, https://commons.wikimedia.org/wiki/File:Conv_layer.png

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

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

and choose

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

Recurrent Neural Net (RNN) = Nonlinear(IIR)

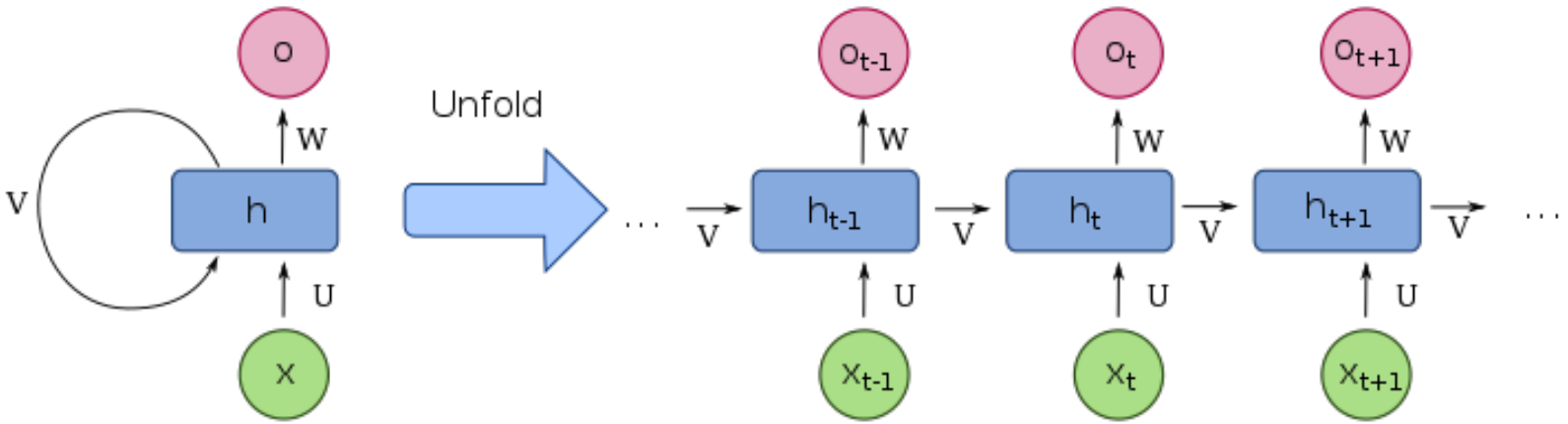


Image CC-SA-4.0 by Ixnay,

`https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg`

Recurrent Neural Net (RNN) = Nonlinear(IIR)

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

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

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

and choose

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

Outline

Partial Derivatives

In order to do back-propagation in recurrent neural networks, it will be important to distinguish between partial and total derivatives. Unfortunately, these are not defined very clearly in introductory calculus classes.
The standard definition of the partial derivative of $f(\vec{x})$ w.r.t. x_1, where $\vec{x} = [x_1, \ldots, x_D]^T$, is

$$\frac{\partial f}{\partial x_1} = \lim_{\epsilon \to 0} \left(\frac{f(x_1 + \epsilon, x_2, \ldots) - f(x_1, x_2, \ldots)}{\epsilon} \right)$$

Partial Derivatives

$$\frac{\partial f}{\partial x_1} = \lim_{\epsilon \to 0} \left(\frac{f(x_1 + \epsilon, x_2, \ldots) - f(x_1, x_2, \ldots)}{\epsilon} \right)$$

In other words, $\frac{\partial f}{\partial x_k}$ is defined as the derivative of f w.r.t. x_k while holding all of the other x_d, for $1 \leq d \leq D$, constant.

Total Derivatives

The partial derivative and total derivative differ if some of the **other** elements of the vector $\vec{x}$ might depend on x_k. For example, suppose that each x_j is a function of x_i for $i \leq j$:

$$x_j = g_j(x_1, \ldots, x_{j-1})$$

Then the **total** derivative allows each of the x_j, for $j > k$, to vary as x_k varies:

$$\frac{df}{dx_1} = \lim_{\epsilon \to 0} \left(\frac{f(x_1 + \epsilon, x_2(x_1 + \epsilon), \ldots) - f(x_1, x_2(x_1), \ldots)}{\epsilon} \right)$$

Partial and Total Derivatives

- The **partial derivative** of f w.r.t. x_k holds all of the other variables constant, while varying **only** x_k. The other variables are held constant **ignoring any dependency they otherwise would have on** x_k:

$$\frac{\partial f}{\partial x_1} = \lim_{\epsilon \to 0} \left(\frac{f(x_1 + \epsilon, x_2(x_1), \ldots) - f(x_1, x_2(x_1), \ldots)}{\epsilon} \right)$$

- The **total derivative** takes into account the effect that varying x_k might have on all the other variables:

$$\frac{df}{dx_1} = \lim_{\epsilon \to 0} \left(\frac{f(x_1 + \epsilon, x_2(x_1 + \epsilon), \ldots) - f(x_1, x_2(x_1), \ldots)}{\epsilon} \right)$$

Partial and Total Derivatives

- So far, we've pretended that, when we say "holding all other variables constant," we know what that means.
- In a neural network, there are lots of implicit variables, that you can calculate if you want to.
- When you say "holding all other variables constant," it is necessary to specify exactly which other variables you mean.

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

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

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

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

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

$$\frac{d\mathcal{L}}{dw_{lk}} = \epsilon_l h_k$$

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

$$\epsilon_l = \frac{d\mathcal{L}}{de_l}$$

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 $\hat{y}$:

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

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

Then the back-prop gradients are the derivatives of $\mathcal{L}$ with respect to the **excitations** at each node:

$$\frac{d\mathcal{L}}{dw_{lk}^{(2)}} = \epsilon_l h_k, \quad \epsilon_l = \frac{d\mathcal{L}}{d\xi_l^{(2)}}$$

$$\frac{d\mathcal{L}}{dw_{kj}^{(1)}} = \delta_k x_j, \quad \delta_k = \frac{d\mathcal{L}}{d\xi_k^{(1)}}$$

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

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

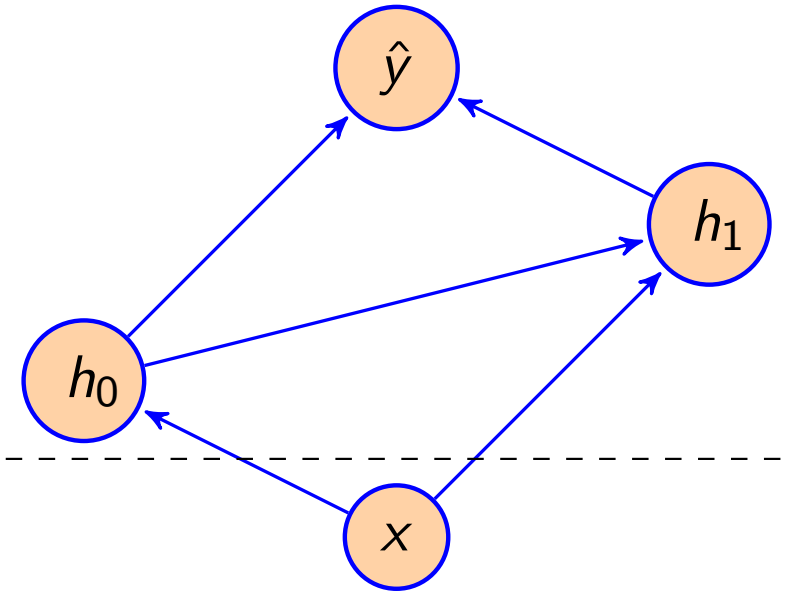
Back-Prop Example

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?

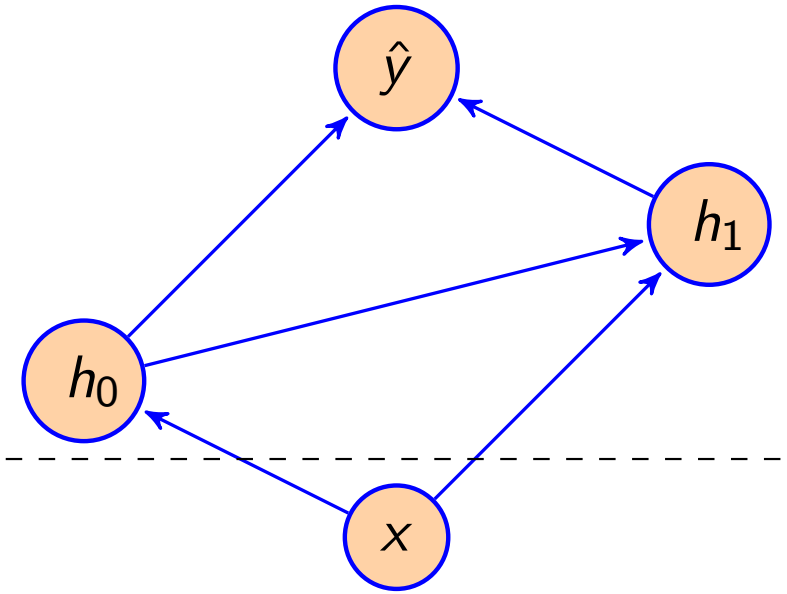
Flow Graphs



We often show the flow 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 flow 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.

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

Flow Graphs



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

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.

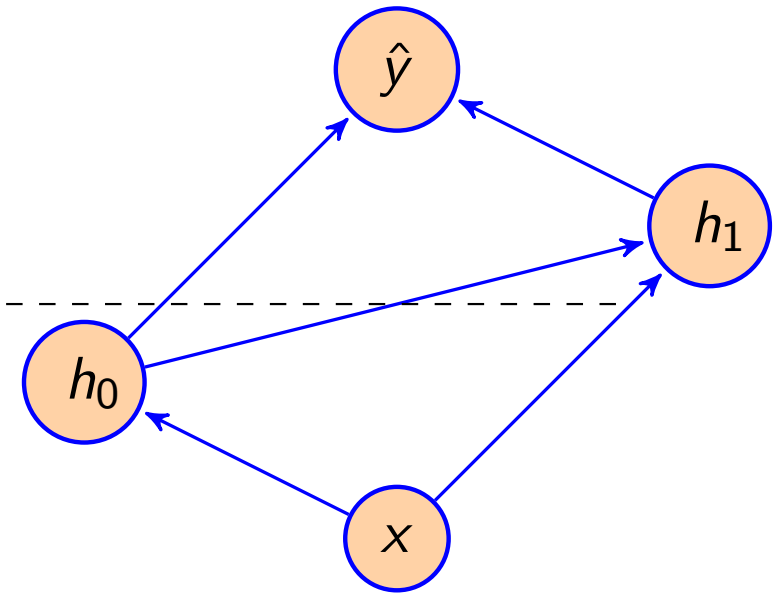
Back-Prop Example

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

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

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

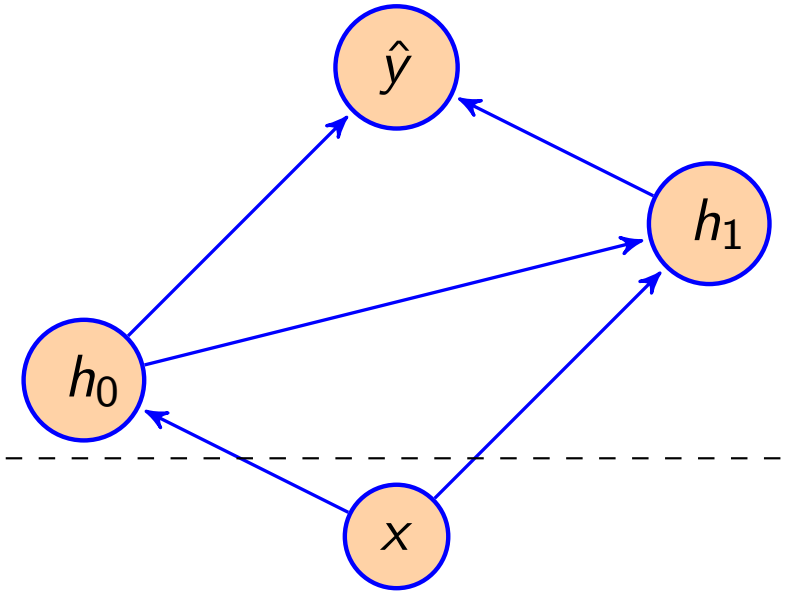
Back-Prop Example



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 $\frac{d\hat{y}}{dx}$:

$$\frac{d\hat{y}}{dx} = \frac{d\hat{y}}{dh_1}\frac{\partial h_1}{\partial x} + \frac{d\hat{y}}{dh_0}\frac{\partial h_0}{\partial x}$$

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

Back-Prop and Partial Derivativees

Suppose we have a neural net, defined by

$$\xi_k^{(l+1)} = \sum_j w_{k,j}^{(l+1)} \sigma\left(\xi_j^{(l)}\right)$$

then, if we define

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}}{\partial \xi_j^{(l)}}$$

we can compute it as

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} \left(\frac{\partial \xi_k^{(l+1)}}{\partial \xi_j^{(l)}} \right)$$

Here, the partial-derivative sign ∂ means that we hold constant all other ξ variables **at the same layer**. We're obviously not holding constant all excitations at the **next** layer, because those are the things over which we compute the back-prop.

Outline

Conclusions

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

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

- 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.

Outline

Written Example

Consider a set of variables (u, v, w, x, y, z) with the following relationships:

$$
\begin{aligned}
u &= \epsilon_u \\
v &= 0.1u + \epsilon_v \\
w &= 0.1v + 0.1u + \epsilon_w \\
x &= 0.1w + 0.1v + 0.1u + \epsilon_x \\
y &= 0.1x + 0.1w + 0.1v + 0.1u + \epsilon_y \\
z &= 0.1y + 0.1x + 0.1w + 0.1v + 0.1u + \epsilon_z
\end{aligned}
$$

1. Draw a flow-graph.
2. Calculate the gradient of z w.r.t. the vector $\vec{\phi} = [u, v]^T$.