Lecture 17: Recurrent Neural Nets

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing

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 design a frequency response:

$$H(\omega) = \sum_{n=0}^{N-1} h[n]e^{-j\omega n}$$

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 design a frequency response: The coefficients, $h[m]$, are chosen in order to design a frequency response:

$$H(\omega) = \frac{\sum_{m=0}^{N-1} b_m e^{-j\omega m}}{1 - \sum_{m=1}^{M-1} a_m e^{-j\omega m}}$$

Outline

Convolutional Neural Net = Nonlinear(FIR)

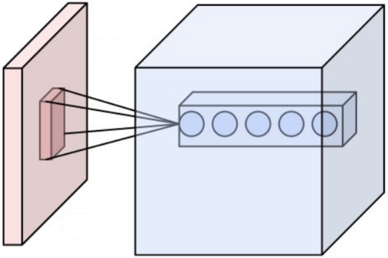


Image CC-SA-4.0 by Aphex34, `https://commons.wikimedia.org/wiki/File:Conv_layer.png`

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

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

and choose

$$w[n] \leftarrow w[n] - \eta\frac{\partial\mathcal{L}}{\partial w[n]}$$

Recurrent Neural Net (RNN) = Nonlinear(IIR)

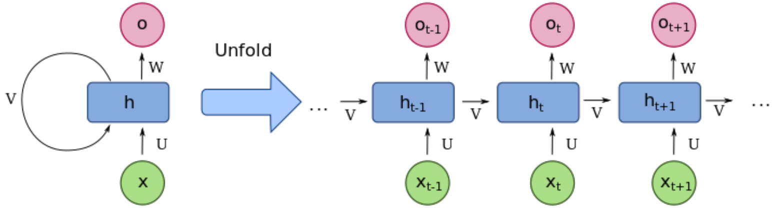


Image CC-SA-4.0 by Ixnay,

`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

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

and choose

$$w[m] \leftarrow w[m] - \eta\frac{\partial\mathcal{L}}{\partial w[m]}$$

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 $1 \le d \le D$, 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 $i \le j$:

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

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:

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

The Chain Rule

Suppose that f depends on x, and also on a number of intermediate variables $z_1,\ldots,z_N$. Suppose, too, that z_1 depends on $z_2,\ldots,z_N$, and so on. Then:

$$\begin{aligned}\frac{df}{dx} &= \frac{\partial f}{\partial x} + \sum_{i=1}^{N} \frac{df}{dz_i}\frac{\partial z_i}{\partial x}\\ &= \frac{\partial f}{\partial x} + \sum_{i=1}^{N} \frac{\partial f}{\partial z_i}\frac{dz_i}{dx}\end{aligned}$$

Notice that either the $\frac{df}{dz_i}$ are total, or the $\frac{dz_i}{dx}$ are total, but not both. You need to choose: will you model all of the interactions in the second half (the $\frac{df}{dz_i}$), or in the first half (the $\frac{dz_i}{dx}$)? Either one is fine.

Chain Rule 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

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

Chain Rule Example

Suppose we have the following network:

$$
\begin{aligned}
h_0 &= \cos(x) \\
h_1 &= \frac{1}{\sqrt{2}}\left(h_0^3 + \sin(x)\right) \\
\hat{y} &= \sqrt{h_0^2 + h_1^2}
\end{aligned}
$$

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

Back-Prop Example

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

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

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

Chain Rule Example

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

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

Chain Rule Example

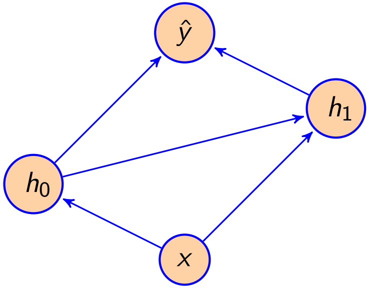
Third, back-prop to find $\frac{d\hat{y}}{dx}$:

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

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

Outline

Flow Graphs


$$\begin{tikzcd} & \hat{y} & \\ & & h_1 \arrow[ul] \\ h_0 \arrow[uur] \arrow[urr] & & \\ & x \arrow[ul] \arrow[uur] & \end{tikzcd}$$

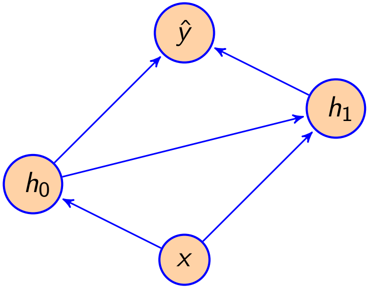
Forward propagation can be summarized by a **flow graph**, which specifies the dependencies among variables, without specifying the functional form of the dependence. For example, the above graph shows that

- $\hat{y}$ is a function of h_0 and h_1.
- h_1 is a function of x and h_0.
- h_0 is a function of x.

Review: Partial and Total Derivatives

- The **total derivative** symbol, $\frac{d\mathcal{L}}{dh_k}$, **always means the same thing**: derivative including the contributions of all paths from h_k to $\mathcal{L}$.
- The **partial derivative** symbol, $\frac{\partial\mathcal{L}}{\partial h_k}$, can mean **different things in different equations** (because different equations might hold constant a different set of other variables).
- There is a notation we can use to specify **which** other variables are being held constant: $\frac{\partial\mathcal{L}}{\partial h_k}(\hat{y}_1, \hat{y}_6, \hat{y}_{10}, h_1, \ldots, h_N)$ means "hold $\hat{y}_1, \hat{y}_6, \hat{y}_{10}$, and $h_1, \ldots, h_{k-1}, h_{k+1}, \ldots, h_N$ constant."

Chain Rule: Total Derivative



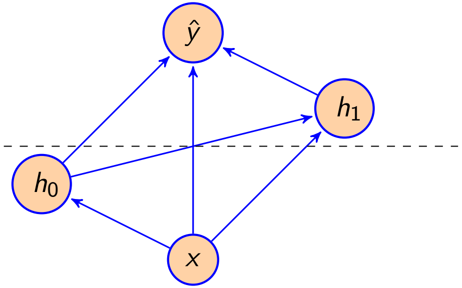
Suppose we want to find $\frac{d\hat{y}}{dx}$. The total derivative can be computed using the chain rule:

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

where

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

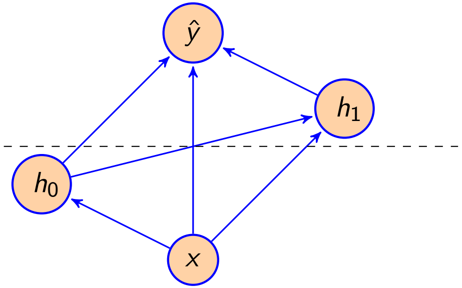
Chain Rule: Partial Derivatives



A far more common thing to do in neural nets, though, is to find the partial derivative while holding only **some** of the other variables constant. For instance, as suggested by the dashed line above, suppose we want to find the partial derivative $\frac{\partial \hat{y}}{\partial h_0}$ while holding x constant, but allowing h_1 to vary:

$$\frac{\partial \hat{y}}{\partial h_0}(h_0, x) = ?$$

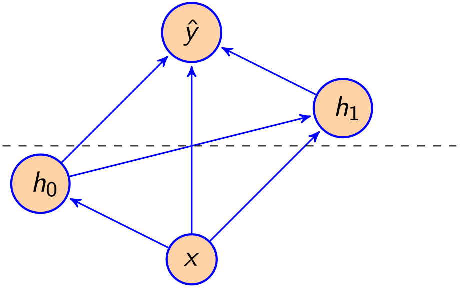
Chain Rule: Partial Derivatives



The chain rule to compute a partial derivative is similar to the one for total derivatives:

$$\frac{\partial \hat{y}}{\partial h_0}(x, h_0) = \frac{\partial \hat{y}}{\partial h_1}(x, h_0, h_1)\frac{\partial h_1}{\partial h_0}(x, h_0, h_1) + \frac{\partial \hat{y}}{\partial h_0}(x, h_0, h_1)$$

Chain Rule: Partial Derivatives



- We start with the partials that hold all other variables frozen, e.g., $\frac{\partial \hat{y}}{\partial h_0}(x, h_0, h_1)$.
- Then we eliminate h_1 from the list of frozen variables, by adding its dependency into all other partials:

$$\frac{\partial \hat{y}}{\partial h_0}(x, h_0) = \frac{\partial \hat{y}}{\partial h_1}(x, h_0, h_1)\frac{\partial h_1}{\partial h_0}(x, h_0, h_1) + \frac{\partial \hat{y}}{\partial h_0}(x, h_0, h_1)$$

Outline

Review: Excitation and Activation

- The **activation** of a hidden node is the output of the nonlinearity. For example, in a fully-connected network with outputs $\hat{y}_k$, weights $w_{j,k}^{(l)}$ in the l^{th} layer, bias $b_k^{(l)}$, nonlinearity $g()$, and hidden node activations $\boldsymbol{h}$, the activation of the k^{th} output node is

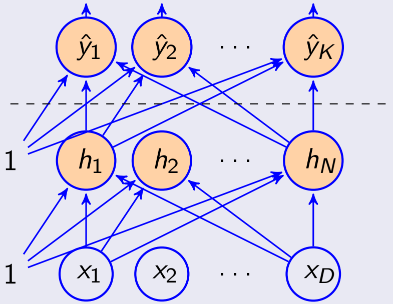
$$\hat{y}_k = g\left(b_k^{(2)} + \sum_{j=1}^{p} w_{k,j}^{(2)} h_j\right)$$

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

$$\xi_k^{(2)} = b_k^{(2)} + \sum_{j=1}^{p} w_{k,j}^{(2)} h_j$$

Fully-Connected Network

$$\hat{y} = h(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)})$$



In a fully-connected network, we usually want two types of derivatives:

- The gradient of $\mathcal{L}$ with respect to a layer, i.e.,

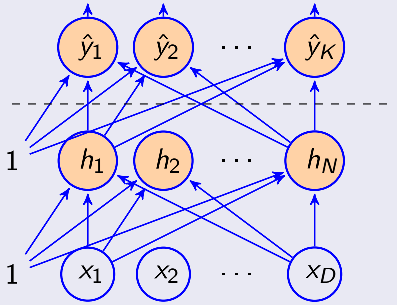
$$\frac{\partial \mathcal{L}}{\partial h_j}(h_1, \ldots, h_N)$$

- The gradient of $\mathcal{L}$ with respect to the network weights, i.e.,

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}^{(1)}}(w_{1,1}^{(1)}, \ldots, w_{K,N}^{(2)})$$

Fully-Connected Network

$$\hat{y} = h(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)})$$



Chain Rule in a Fully-Connected Network

$$\frac{\partial \mathcal{L}}{\partial h_j}(\boldsymbol{h}) = \sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial \hat{y}_k}(\hat{\boldsymbol{y}}, \boldsymbol{h}) \times \frac{\partial \hat{y}_k}{\partial h_j}(\hat{\boldsymbol{y}}, \boldsymbol{h})$$

New Notation

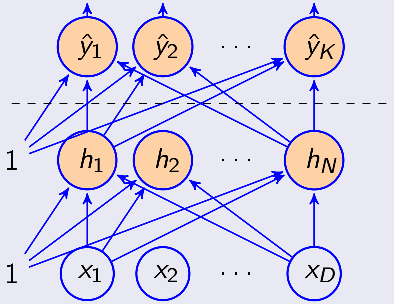
The notation on the previous slide is cumbersome. Instead, let's use:

- $\frac{\partial \hat{y}_k}{\partial x_j}$ means "the derivative of $\hat{y}_k$ with respect to x_j if **all** other variables are held constant." In this graph, for example, $\frac{\partial \hat{y}_k}{\partial x_j} = 0$.
- $\frac{d\hat{y}_k}{dx_j}$ means "the derivative of $\hat{y}_k$ with respect to x_j if the variables on the path between them are allowed to vary."
- In many cases, those two will be the same, e.g., in this graph,

$$\begin{aligned}\frac{d\hat{y}_k}{dx_j} &= \sum_i \frac{\partial \hat{y}_k}{\partial h_i}\frac{\partial h_i}{\partial x_j} \\ &= \sum_i \frac{d\hat{y}_k}{dh_i}\frac{dh_i}{dx_j}\end{aligned}$$

Fully-Connected Network

$$\hat{y} = h(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)})$$



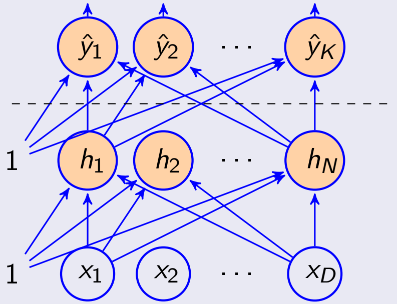
New Notation

Then we can write:

$$\frac{d\mathcal{L}}{dh_j} = \sum_{k=1}^{K} \frac{d\mathcal{L}}{d\hat{y}_k} \frac{\partial \hat{y}_k}{\partial h_j}$$

Fully-Connected Network

$$\hat{y} = h(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)})$$



Back-Prop in a Fully-Connected Network

Similarly,

$$\frac{d\mathcal{L}}{dw_{j,k}^{(1)}} = \sum_{j=1}^{N} \frac{d\mathcal{L}}{dh_j} \frac{\partial h_j}{\partial w_{j,k}^{(1)}}$$
$$= \sum_{j=1}^{N} \frac{d\mathcal{L}}{dh_j} x_k$$

Gradients and Partial Derivatives

A warning about our notation:

- The gradient is defined to be a vector of partial derivatives, i.e.,

$$\nabla_{\boldsymbol{h}}\mathcal{L} \equiv \left[\begin{array}{c} \frac{\partial\mathcal{L}}{\partial h_1}(h_1,\ldots,h_N) \\ \vdots \\ \frac{\partial\mathcal{L}}{\partial h_N}(h_1,\ldots,h_N) \end{array}\right]$$

- The partial derivative only keeps constant the other elements of $\boldsymbol{h}$—it does not keep constant any other variables on the path between $\boldsymbol{h}$ and $\mathcal{L}$. Thus, usually, we can write

$$\nabla_{\boldsymbol{h}}\mathcal{L} \equiv \left[\begin{array}{c} \frac{\partial\mathcal{L}}{\partial h_1}(h_1,\ldots,h_N) \\ \vdots \\ \frac{\partial\mathcal{L}}{\partial h_N}(h_1,\ldots,h_N) \end{array}\right] = \left[\begin{array}{c} \frac{d\mathcal{L}}{dh_1} \\ \vdots \\ \frac{d\mathcal{L}}{dh_N} \end{array}\right]$$

Outline

Back-Prop in an RNN

Suppose we have a recurrent neural net, defined by

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

then

$$\frac{d\mathcal{L}}{dw[m]} = \sum_n \frac{d\mathcal{L}}{d\xi[n]}\frac{\partial \xi[n]}{\partial w[m]}$$
$$= \sum_n \frac{d\mathcal{L}}{d\xi[n]} h[n-m]$$

Partial vs. Full Derivatives

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

$$\mathcal{L} = \frac{1}{2}\sum_n \left(h[n] - y[n]\right)^2$$

Notice that $\mathcal{L}$ depends on $h[n]$ in many different ways:

$$\frac{d\mathcal{L}}{dh[n]} = \frac{\partial\mathcal{L}}{\partial h[n]} + \frac{d\mathcal{L}}{dh[n+1]}\frac{\partial h[n+1]}{\partial h[n]} + \frac{d\mathcal{L}}{dh[n+2]}\frac{\partial h[n+2]}{\partial h[n]} + \ldots$$

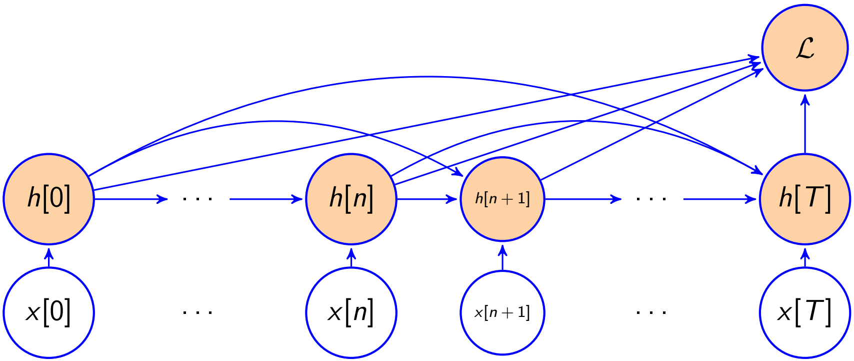
Partial vs. Full Derivatives

In general,

$$\frac{d\mathcal{L}}{dh[n]} = \frac{\partial\mathcal{L}}{\partial h[n]} + \sum_{m=1}^{\infty} \frac{d\mathcal{L}}{dh[n+m]} \frac{\partial h[n+m]}{\partial h[n]}$$

where

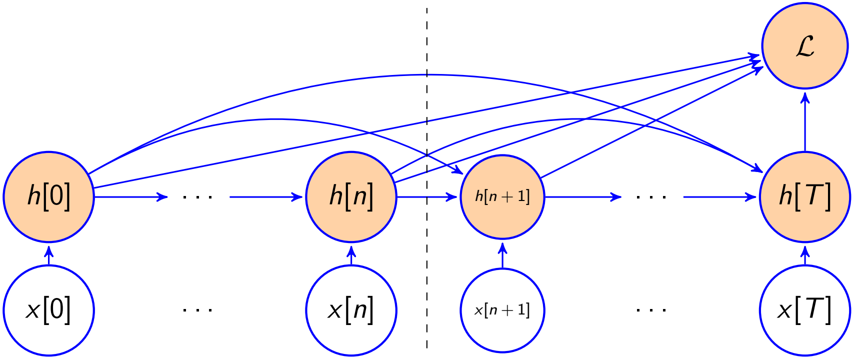
- $\frac{d\mathcal{L}}{dh[n]}$ includes all of the different ways in which a change of $h[n]$ might effect $\mathcal{L}$.
- $\frac{\partial h[n+m]}{\partial h[n]}$ keeps constant all variables on the path between $h[n]$ and $h[n+m]$. It only measures the direct connection between these two variables.



Here's a flow diagram that could represent:

$$h[n] = g\left(x[n] + \sum_{m=0}^{\infty} w[m]h[n-m]\right)$$

$$\mathcal{L} = \frac{1}{2}\sum_n (y[n] - h[n])^2$$



Back-propagation through time does this:

$$\frac{d\mathcal{L}}{dh[n]} = \frac{\partial \mathcal{L}}{\partial h[n]} + \sum_{m=1}^{T-n} \frac{d\mathcal{L}}{dh[n+m]} \frac{\partial h[n+m]}{\partial h[n]}$$

Partial vs. Total Derivatives

So for example, if

$$\mathcal{L} = \frac{1}{2}\sum_n (h[n] - y[n])^2$$

then the partial derivative of $\mathcal{L}$ w.r.t. $h[n]$ while keeping all other variables constant is:

$$\frac{\partial \mathcal{L}}{\partial h[n]} = h[n] - y[n]$$

and the total derivative of $\mathcal{L}$ w.r.t. $h[n]$ is

$$\frac{d\mathcal{L}}{dh[n]} = (h[n] - y[n]) + \sum_{m=1}^{\infty} \frac{d\mathcal{L}}{dh[n+m]} \frac{\partial h[n+m]}{\partial h[n]}$$

Partial vs. Total Derivatives

So for example, if

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

then the partial derivative of $h[n+k]$ w.r.t. $h[n]$ is

$$\frac{\partial h[n+k]}{\partial h[n]} = \dot{g}(\xi[n+k])w[k]$$

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

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 $\mathcal{L}$ w.r.t. the excitation $\xi[n]$ at time n, assuming that all other time steps are held constant.

$$\frac{\partial \mathcal{L}}{\partial \xi[n]}$$

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

$$\frac{d\mathcal{L}}{d\xi[n]}$$

Synchronous Backprop in an RNN

Suppose we have a recurrent neural net, defined by

$$\begin{aligned}
\xi[n] &= x[n] + \sum_{m=1}^{M} w[m]h[n-m] \\
h[n] &= g\left(\xi[n]\right) \\
\mathcal{L} &= \frac{1}{2}\sum_{n}\left(h[n]-y[n]\right)^2
\end{aligned}$$

then

$$\frac{\partial \mathcal{L}}{\partial h[n]} = \left(h[n]-y[n]\right)$$

Back-Prop Through Time (BPTT)

Suppose we have a recurrent neural net, defined by

$$\xi[n] = x[n] + \sum_{m=1}^{M} w[m]h[n-m]$$
$$h[n] = g\left(\xi[n]\right)$$
$$\mathcal{L} = \frac{1}{2}\sum_{n}\left(h[n]-y[n]\right)^2$$

then

$$\frac{d\mathcal{L}}{dh[n]} = \frac{\partial\mathcal{L}}{\partial h[n]} + \sum_{m=1}^{\infty}\frac{d\mathcal{L}}{dh[n+m]}\frac{\partial h[n+m]}{\partial h[n]}$$
$$= \frac{\partial\mathcal{L}}{\partial h[n]} + \sum_{m=1}^{M}\frac{d\mathcal{L}}{dh[n+m]}\dot{g}(\xi[n+m])w[m]$$

Weight Gradient

Suppose we have a recurrent neural net, defined by

$$\xi[n] = x[n] + \sum_{m=1}^{M} w[m]h[n-m]$$
$$h[n] = g\left(\xi[n]\right)$$
$$\mathcal{L} = \frac{1}{2}\sum_{n}\left(h[n]-y[n]\right)^2$$

then the weight gradient is given by

$$\frac{\partial \mathcal{L}}{\partial w[m]}\left(w[1],\ldots,w[M]\right) = \sum_{n} \frac{d\mathcal{L}}{dh[n]}\frac{\partial h[n]}{\partial w[m]}\left(w[1],\ldots,w[M]\right)$$
$$= \sum_{n} \frac{d\mathcal{L}}{dh[n]}\dot{g}(\xi[n])h[n-m]$$

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.

$$\frac{d\mathcal{L}}{dh[n]} = \frac{\partial\mathcal{L}}{\partial h[n]} + \sum_{m=1}^{M} \frac{d\mathcal{L}}{dh[n+m]}\dot{g}(\xi[n+m])w[m]$$

$$\frac{\partial\mathcal{L}}{\partial w[m]}(w[1],\ldots,w[M]) = \sum_{n} \frac{d\mathcal{L}}{dh[n]}\dot{g}(\xi[n])h[n-m]$$

Outline

Written Example

Suppose that $\boldsymbol{h}[t] = [h_1[t], \ldots, h_N[t]]^T$ is a vector, and suppose that

$$\boldsymbol{h}[t] = \tanh\left(\boldsymbol{U}\boldsymbol{x}[t] + \boldsymbol{V}_1\boldsymbol{h}[t-1] + \boldsymbol{V}_2\boldsymbol{h}[t-2]\right)$$

$$\mathcal{L} = \frac{1}{2}\sum_t \|\boldsymbol{y} - \boldsymbol{W}\boldsymbol{h}[t]\|^2$$

where $\boldsymbol{U}$ is a $N \times D$ matrix, $\boldsymbol{W}$ is a $K \times N$ matrix, and $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ are $N \times N$ matrices. Find an algorithm to compute $\nabla_{\boldsymbol{h}[t]}\mathcal{L}$.