Lecture 23: Recurrent Neural Nets

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing

Outline

Convolutional Neural Net = Nonlinear(FIR)

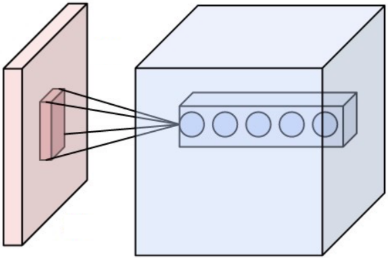


Image CC-SA-4.0 by Aphex34, https://commons.wikimedia.org/wiki/File:Conv_layer.png

Convolutional Neural Net = Nonlinear(FIR)

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

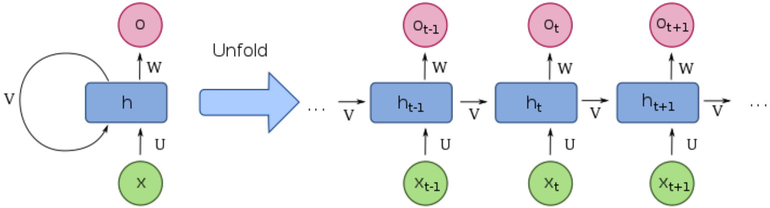
The coefficients, $w[m]$, are chosen to minimize some kind of loss function. For example, suppose that the goal is to make $h[n]$ resemble a target signal $y[n]$; then we might use

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

and choose

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

Recurrent Neural Net (RNN) = Nonlinear(IIR)



Recurrent Neural Net (RNN) = Nonlinear(IIR)

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

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

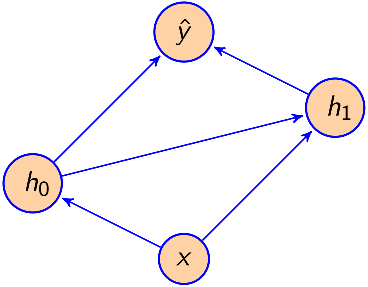
$$\mathcal{L} = \frac{1}{2}\sum_{n=0}^{N}(h[n]-y[n])^2$$

and choose

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

Outline

Flow Graphs



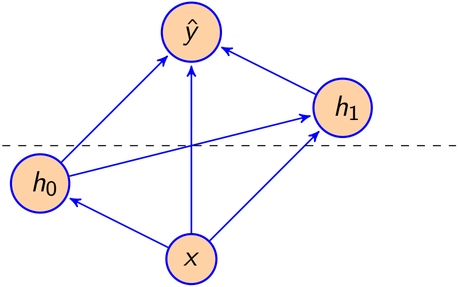
Forward propagation can be summarized by a **flow graph**, which specifies the dependencies among variables, without specifying the functional form of the dependence. For example, the above graph shows that

- $\hat{y}$ 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."

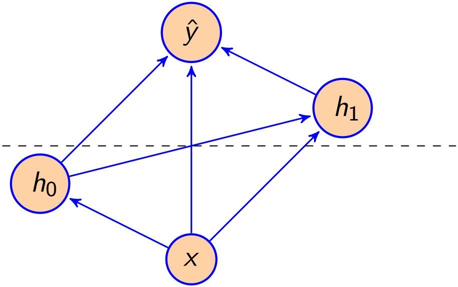
Back-Propagation in terms of Flow Graphs



In order to find the derivative of an output w.r.t. any intermediate variables, one strategy that works is:

1. Draw a dashed line across the graph just downstream of the desired intermediate variables.
2. Apply the chain rule, with a summation across all edges that cross the dashed line.

Back-Propagation in terms of Flow Graphs

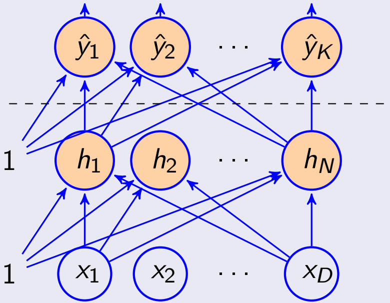


$$\frac{\partial \hat{y}}{\partial h_0}(x, h_0) = \frac{d\hat{y}}{dh_1}\frac{\partial h_1}{\partial h_0}(x, h_0, h_1) + \frac{d\hat{y}}{d\hat{y}}\frac{\partial \hat{y}}{\partial h_0}(x, h_0, h_1)$$
$$\frac{\partial \hat{y}}{\partial x}(x, h_0) = \frac{d\hat{y}}{dh_1}\frac{\partial h_1}{\partial x}(x,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

Notice: $\frac{\partial \hat{y}}{\partial x}(x, h_0)$ does **not** include $\frac{d\hat{y}}{dh_0}\frac{\partial h_0}{\partial x}$.

Fully-Connected Network

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

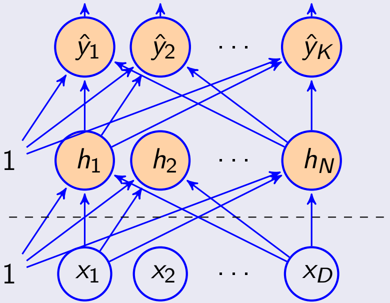


Back-Prop in a Fully-Connected Network

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

Fully-Connected Network

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



Back-Prop in a Fully-Connected Network

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

Outline

Back-Prop in a CNN

Suppose we have a convolutional neural net, defined by

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

then

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

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

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w[m]} &= \sum_n \frac{d\mathcal{L}}{d\xi[n]} \frac{\partial \xi[n]}{\partial w[m]} \\ &= \sum_n \frac{d\mathcal{L}}{d\xi[n]} h[n-m]\end{aligned}$$

Outline

Partial vs. Full Derivatives

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

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

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

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

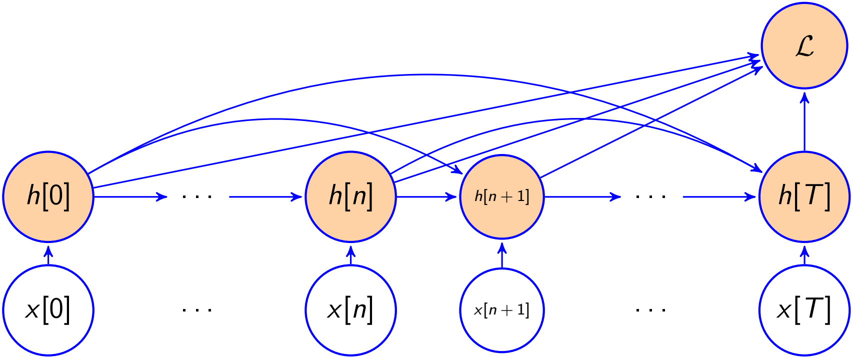
Partial vs. Full Derivatives

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

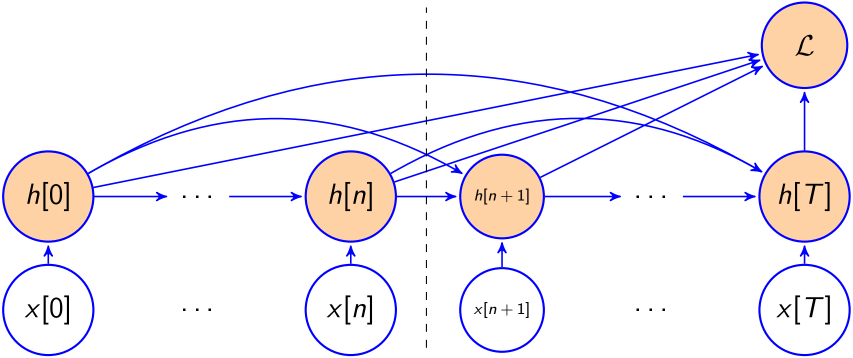
- $\frac{d\mathcal{L}}{dh[n]}$ is the total derivative, and includes all of the different ways in which $\mathcal{L}$ depends on $h[n]$.
- $\frac{\partial h[n+m]}{\partial h[n]}$ is the partial derivative, i.e., the change in $h[n+m]$ per unit change in $h[n]$ if $\{h[n+1], \ldots, h[n+m-1]\}$ are all held constant.



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. Full 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. $h[n]$ 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. Full 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.

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

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

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

Synchronous Backprop in an RNN

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

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

Back-Prop Through Time (BPTT)

Suppose we have a recurrent neural net, defined by

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

then

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

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

$$\vec{h}[t] = \tanh\left(U\vec{x}[t] + V_1\vec{h}[t-1] + V_2\vec{h}[t-2]\right)$$
$$\mathcal{L} = \frac{1}{2}\sum_t \|\vec{y} - W\vec{h}[t]\|^2$$

where U is a $N \times D$ matrix, W is a $K \times N$ matrix, and V_1 and V_2 are $N \times N$ matrices. Find an algorithm to compute $\nabla_{\vec{h}[t]}\mathcal{L}$.