Recurrent Neural Nets

ECE 417: Multimedia Signal Processing
Mark Hasegawa-Johnson

University of Illinois

November 19, 2019
1. Linear Time Invariant Filtering: FIR & IIR
2. Nonlinear Time Invariant Filtering: CNN & RNN
3. Back-Propagation Training for CNN and RNN
4. Back-Prop Through Time
5. Vanishing/Exploding Gradient
6. Gated Recurrent Units
7. Long Short-Term Memory (LSTM)
8. Conclusion
Outline

1. Linear Time Invariant Filtering: FIR & IIR
2. Nonlinear Time Invariant Filtering: CNN & RNN
3. Back-Propagation Training for CNN and RNN
4. Back-Prop Through Time
5. Vanishing/Exploding Gradient
6. Gated Recurrent Units
7. Long Short-Term Memory (LSTM)
8. Conclusion
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:

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

\[ H(z) = \frac{\sum_{m=0}^{N-1} b_m z^{-m}}{1 - \sum_{m=1}^{M-1} a_m z^{-m}} = b_0 \frac{\prod_{k=1}^{N-1} (1 - r_k z^{-1})}{\prod_{k=1}^{M-1} (1 - p_k z^{-1})} \]
## Outline

1. **Linear Time Invariant Filtering: FIR & IIR**
2. **Nonlinear Time Invariant Filtering: CNN & RNN**
3. Back-Propagation Training for CNN and RNN
4. Back-Prop Through Time
5. Vanishing/Exploding Gradient
6. Gated Recurrent Units
7. Long Short-Term Memory (LSTM)
8. Conclusion
Convolutional Neural Net = Nonlinear(FIR)

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

The coefficients, \( h[m] \), are chosen to minimize some kind of error. For example, suppose that the goal is to make \( y[n] \) resemble a target signal \( t[n] \); then we might use

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

and choose

\[ h[n] \leftarrow h[n] - \eta \frac{dE}{dh[n]} \]
Recurrent Neural Net (RNN) = Nonlinear(IIR)

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

The coefficients, \( a_m \), are chosen to minimize the error. For example, suppose that the goal is to make \( y[n] \) resemble a target signal \( t[n] \); then we might use

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

and choose

\[ a_m \leftarrow a_m - \eta \frac{dE}{da_m} \]
Outline

1. Linear Time Invariant Filtering: FIR & IIR
2. Nonlinear Time Invariant Filtering: CNN & RNN
3. Back-Propagation Training for CNN and RNN
4. Back-Prop Through Time
5. Vanishing/Exploding Gradient
6. Gated Recurrent Units
7. Long Short-Term Memory (LSTM)
8. Conclusion
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 $z_l$, weights $\vec{v}$, bias $v_0$, nonlinearity $g()$, and hidden node activations $\vec{y}$, the activation of the $l^{th}$ output node is

$$z_l = g \left( v_{l0} + \sum_{k=1}^{p} v_{lk} y_k \right)$$

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

$$e_l = v_{l0} + \sum_{k=1}^{p} v_{lk} y_k$$
The **excitation** of a hidden node is the input of the nonlinearity. For example, the excitation of the node above is

\[ e_l = v_{l0} + \sum_{k=1}^{p} v_{lk} y_k \]

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

\[ \frac{dE}{dv_{lk}} = \epsilon_l y_k \]

where \( \epsilon_l \) is the derivative of the error w.r.t. the \( l^{th} \) excitation:

\[ \epsilon_l = \frac{dE}{de_l} \]
Suppose we have a fully-connected network, with inputs $\vec{x}$, weight matrices $U$ and $V$, nonlinearities $g()$ and $h()$, and output $z$:

$$e_k = u_{k0} + \sum_j u_{kj}x_j$$

$$y_k = g(e_k)$$

$$e_l = v_{l0} + \sum_k v_{lk}y_k$$

$$z_l = h(e_l)$$

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

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

$$\delta_k = \frac{dE}{de_k}$$
Back-Prop in a CNN

Suppose we have a convolutional neural net, defined by

\[ e[n] = \sum_{m=0}^{N-1} h[m] \times [n - m] \]

\[ y[n] = g(e[n]) \]

then

\[ \frac{dE}{dh[m]} = \sum_{n} \delta[n] \times [n - m] \]

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

\[ e[n] = x[n] + \sum_{m=1}^{M-1} a_m y[n - m] \]

\[ y[n] = g(e[n]) \]

then

\[ \frac{dE}{da_m} = \sum_n \delta[n] y[n - m] \]

where \( y[n - m] \) is calculated by forward-propagation, and then \( \delta[n] \) is calculated by back-propagation as

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

1. Linear Time Invariant Filtering: FIR & IIR
2. Nonlinear Time Invariant Filtering: CNN & RNN
3. Back-Propagation Training for CNN and RNN
4. Back-Prop Through Time
5. Vanishing/Exploding Gradient
6. Gated Recurrent Units
7. Long Short-Term Memory (LSTM)
8. Conclusion
Partial vs. Full Derivatives

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

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

Notice that $E$ depends on $y[n]$ in many different ways:

$$\frac{dE}{dy[n]} = \frac{\partial E}{\partial y[n]} + \frac{dE}{dy[n+1]} \frac{\partial y[n+1]}{\partial y[n]} + \frac{dE}{dy[n+2]} \frac{\partial y[n+2]}{\partial y[n]} + \ldots$$
In general, 

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

where

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

So for example, if

\[ E = \frac{1}{2} \sum_n (y[n] - t[n])^2 \]

then the partial derivative of \( E \) w.r.t. \( y[n] \) is

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

and the total derivative of \( E \) w.r.t. \( y[n] \) is

\[ \frac{dE}{dy[n]} = (y[n] - t[n]) + \sum_{m=1}^{\infty} \frac{dE}{dy[n + m]} \frac{\partial y[n + m]}{\partial y[n]} \]
Partial vs. Full Derivatives

So for example, if

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

then the partial derivative of \( y[n+k] \) w.r.t. \( y[n] \) is

\[ \frac{\partial y[n+k]}{\partial y[n]} = a_k \dot{g} \left( x[n+k] + \sum_{m=1}^{M-1} a_m y[n+k-m] \right) \]

where \( \dot{g}(x) = \frac{dg}{dx} \) is the derivative of the nonlinearity. The total derivative of \( y[n+k] \) w.r.t. \( y[n] \) is

\[ \frac{dy[n+k]}{dy[n]} = \frac{\partial y[n+k]}{\partial y[n]} + \sum_{j=1}^{k-1} \frac{dy[n+k]}{dy[n+j]} \frac{\partial y[n+j]}{\partial y[n]} \]
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**

   \[ \delta[n] = \frac{dE}{de[n]} \]
Suppose we have a recurrent neural net, defined by

\[ e[n] = x[n] + \sum_{m=1}^{M-1} a_m y[n - m] \]

\[ y[n] = g(e[n]) \]

\[ E = \frac{1}{2} \sum_n (y[n] - t[n])^2 \]

then

\[ \epsilon[n] = \frac{\partial E}{\partial e[n]} = (y[n] - t[n]) \dot{g}(e[n]) \]

where \( \dot{g}(x) = \frac{dg}{dx} \) is the derivative of the nonlinearity.
Back-Prop Through Time (BPTT)

Suppose we have a recurrent neural net, defined by

\[ e[n] = x[n] + \sum_{m=1}^{M-1} a_m y[n-m] \]

\[ y[n] = g(e[n]) \]

\[ E = \frac{1}{2} \sum_{n} (y[n] - t[n])^2 \]

then

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

\[ = \frac{\partial E}{\partial e[n]} + \sum_{m=1}^{\infty} \frac{dE}{de[n+m]} \frac{\partial e[n+m]}{\partial e[n]} \]

\[ = \epsilon[n] + \sum_{m=1}^{M-1} \delta[n+m] g'(e[n+m]) a_m \]
Outline

1. Linear Time Invariant Filtering: FIR & IIR
2. Nonlinear Time Invariant Filtering: CNN & RNN
3. Back-Propagation Training for CNN and RNN
4. Back-Prop Through Time
5. Vanishing/Exploding Gradient
6. Gated Recurrent Units
7. Long Short-Term Memory (LSTM)
8. Conclusion
Vanishing/Exploding Gradient

- The “vanishing gradient” problem refers to the tendency of \( \frac{dy[n+m]}{de[n]} \) to disappear, exponentially, when \( m \) is large.

- The “exploding gradient” problem refers to the tendency of \( \frac{dy[n+m]}{de[n]} \) to explode toward infinity, exponentially, when \( m \) is large.

- If the largest feedback coefficient is \(|a| > 1\), then you get exploding gradient. If not, you get vanishing gradient.
Example: Vanishing Gradient

Suppose that we have a very simple RNN:

\[ y[n] = bx[n] + ay[n - 1] \]

Suppose that \( x[n] \) is only nonzero at time 0:

\[ x[0] = x_0, \text{ and } x[n] = 0 \forall n \neq 0 \]

Suppose that, instead of measuring \( x[0] \) directly, we are only allowed to measure the output of the RNN \( m \) time-steps later. In order to encourage the neural net to learn \( a \approx 1 \), we might penalize any difference between \( y[m] \) and \( x_0 \), thus:

\[ E = \frac{1}{2} (y[m] - x_0)^2 \]
Example: Vanishing Gradient

Now, how do we perform gradient update of the weights? If

\[ y[n] = bx[n] + ay[n - 1] \]

then

\[
\frac{dE}{db} = \sum_{n} \left( \frac{dE}{dy[n]} \right) x[n]
\]

\[
= \left( \frac{dE}{dy[0]} \right) x[0]
\]

But the error is defined as

\[ E = \frac{1}{2} (y[m] - x_0)^2 \]

so

\[
\frac{dE}{dy[0]} = a \frac{dE}{dy[1]} = a^2 \frac{dE}{dy[2]} = \ldots
\]

\[
= a^m (y[m] - x_0)
\]
Example: Vanishing Gradient

So we find out that the gradient, w.r.t. the coefficient $b$, is either exponentially small, or exponentially large, depending on whether $|a| < 1$ or $|a| > 1$:

$$
\frac{dE}{db} = x_0 (y[m] - x_0) a^m
$$

In other words, if our application requires the neural net to wait $m$ time steps before generating its output, then the gradient is exponentially smaller, and therefore training the neural net is exponentially harder.

Image credit: PeterQ, Wikipedia
Outline

1. Linear Time Invariant Filtering: FIR & IIR
2. Nonlinear Time Invariant Filtering: CNN & RNN
3. Back-Propagation Training for CNN and RNN
4. Back-Prop Through Time
5. Vanishing/Exploding Gradient
6. Gated Recurrent Units
7. Long Short-Term Memory (LSTM)
8. Conclusion
Gated Recurrent Units (GRU)

Gated recurrent units solve the vanishing gradient problem by making the feedback coefficient, $f[n]$, a sigmoidal function of the inputs. When the input causes $f[n] \approx 1$, then the recurrent unit remembers its own past, with no forgetting (no vanishing gradient). When the input causes $f[n] \approx 0$, then the recurrent unit immediately forgets all of the past.

$$y[n] = i[n]x[n] + f[n]y[n - 1]$$

where the input and forget gates depend on $x[n]$ and $y[n]$, as

$$i[n] = \sigma (b_i x[n] + a_i y[n - 1]) \in (0, 1)$$
$$f[n] = \sigma (b_m x[n] + a_f y[n - 1]) \in (0, 1)$$
How does GRU work? Example

For example, suppose that the inputs just coincidentally have values that cause the following gate behavior:

\[ i[n] = \begin{cases} 1 & n = n_0 \\ 0 & \text{otherwise} \end{cases} \]

\[ f[n] = \begin{cases} 0 & n = n_0 \\ 1 & \text{otherwise} \end{cases} \]

\[ y[n] = i[n]x[n] + f[n]y[n-1] \]

Then \( y[N] = y[N-1] = \ldots = y[n_0] = x[n_0] \), memorized! And therefore

\[ \frac{\partial y[N]}{\partial x[n]} = \begin{cases} 1 & n = n_0 \\ 0 & \text{otherwise} \end{cases} \]
Training the Gates

\[ y[n] = i[n]x[n] + f[n]y[n-1] \]
\[ i[n] = \sigma (b_i x[n] + a_i y[n-1]) \in (0,1) \]
\[ f[n] = \sigma (b_m x[n] + a_f y[n-1]) \in (0,1) \]

\[ \frac{\partial E}{\partial b_i} = \sum_{n=0}^{N} \frac{\partial E}{\partial y[n]} \frac{\partial y[n]}{\partial i[n]} \frac{\partial i[n]}{\partial b_i} \]
\[ = \sum_{n=0}^{N} \delta[n]x[n] \frac{\partial i[n]}{\partial b_i} \]
Outline

1. Linear Time Invariant Filtering: FIR & IIR
2. Nonlinear Time Invariant Filtering: CNN & RNN
3. Back-Propagation Training for CNN and RNN
4. Back-Prop Through Time
5. Vanishing/Exploding Gradient
6. Gated Recurrent Units
7. Long Short-Term Memory (LSTM)
8. Conclusion
Characterizing Human Memory

Pr \{\text{remember}\} = p_{LTM}e^{-t/T_{LTM}} + (1 - p_{LTM})e^{-t/T_{STM}}
Neural Network Model: LSTM

\[
i[n] = \text{input gate} = \sigma(b_i x[n] + a_i c[n - 1])
\]
\[
o[n] = \text{output gate} = \sigma(b_o x[n] + a_o c[n - 1])
\]
\[
f[n] = \text{forget gate} = \sigma(b_f x[n] + a_f c[n - 1])
\]
\[
c[n] = \text{memory cell}
\]
\[
y[n] = o[n] c[n]
\]
\[
c[n] = f[n] c[n - 1] + i[n] g(b_c x[n] + a_c c[n - 1])
\]
Outline

1. Linear Time Invariant Filtering: FIR & IIR
2. Nonlinear Time Invariant Filtering: CNN & RNN
3. Back-Propagation Training for CNN and RNN
4. Back-Prop Through Time
5. Vanishing/Exploding Gradient
6. Gated Recurrent Units
7. Long Short-Term Memory (LSTM)
8. Conclusion
TDNN is a one-dimensional ConvNet, the nonlinear version of an FIR filter. Coefficients are shared across time steps.

RNN is 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.

Vanishing gradient problem: the memory of an RNN decays exponentially.

Solution: GRU

An LSTM is a GRU with one more gate, allowing it to decide when to output information from LTM back to STM.