Deep Learning
Recurrent Networks : 1
Fall 2023

Instructor: Bhiksha Raj
Modelling Series

• In many situations one must consider a series of inputs to produce an output
  – Outputs too may be a series

• Examples: ..
What did I say?

“To be” or not “to be”??

• Speech Recognition
  – Analyze a series of spectral vectors, determine what was said
• Note: Inputs are sequences of vectors. Output is a classification result
What is he talking about?

“Football” or “basketball”?

The Steelers, meanwhile, continue to struggle to make stops on defense. They’ve allowed, on average, 30 points a game, and have shown no signs of improving anytime soon.

• Text analysis
  – E.g. analyze document, identify topic
    • Input series of words, output classification output
  – E.g. read English, output French
    • Input series of words, output series of words
Should I invest..

To invest or not to invest?

• Note: Inputs are sequences of vectors. Output may be scalar or vector
  – Should I invest, vs. should I not invest in X?
  – Decision must be taken considering how things have fared over time
These are classification and prediction problems

• Consider a sequence of inputs
  – Input vectors

• Produce one or more outputs

• This can be done with neural networks
  – Obviously
Representational shortcut

- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything by simple boxes
  - Each box actually represents an entire *layer with many units*
Representational shortcut

- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything by simple boxes
  - Each box actually represents an entire *layer with many units*
Representational shortcut

• Input at each time is a vector
• Each layer has many neurons
  – Output layer too may have many neurons
• But will represent everything as simple boxes
  – Each box actually represents an entire layer with many units
The stock prediction problem...

To invest or not to invest?

- Stock market
  - Must consider the series of stock values in the past several days to decide if it is wise to invest today
The stock predictor network

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*
The stock predictor network

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*
The stock predictor network

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*
The stock predictor network

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*
The stock predictor network

• The sliding predictor
  – Look at the last few days
  – This is just a convolutional neural net applied to series data
    • Also called a *Time-Delay neural network*
Finite-response model

• This is a *finite response* system
  – Something that happens *today* only affects the output of the system for \( N \) days into the future
    • \( N \) is the *width* of the system

\[
Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})
\]
The stock predictor

• This is a *finite response* system
  – Something that happens *today* only affects the output of the system for $N$ days into the future
    • $N$ is the *width* of the system
      
      $$Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})$$
The stock predictor

- This is a \textit{finite response} system
  - Something that happens \textit{today} only affects the output of the system for \( N \) days into the future
  - \( N \) is the \textit{width} of the system

\[
Y_t = f(X_t, X_{t-1}, ..., X_{t-N})
\]
The stock predictor

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for *N* days into the future
  - *N* is the *width* of the system

\[ Y_t = f(X_t, X_{t-1}, ..., X_{t-N}) \]
The stock predictor

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for $N$ days into the future
    - $N$ is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, ..., X_{t-N})$$
The stock predictor

• This is a finite response system
  – Something that happens today only affects the output of the system for $N$ days into the future
  • $N$ is the width of the system

$$Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})$$
The stock predictor

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for $N$ days into the future
    - $N$ is the *width* of the system

\[
Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})
\]
Something that happens today only affects the output of the system for $N$ days into the future

- Predictions consider $N$ days of history

To consider more of the past to make predictions, you must increase the “history” considered by the system.
• Problem: Increasing the “history” makes the network more complex
  – No worries, we have the CPU and memory
  • Or do we?
Systems often have long-term dependencies

- Longer-term trends –
  - Weekly trends in the market
  - Monthly trends in the market
  - Annual trends
  - Though longer historic tends to affect us less than more recent events.
We want infinite memory

• Required: *Infinite* response systems
  – What happens today can continue to affect the output forever
    • Possibly with weaker and weaker influence
      \[ Y_t = f(X_t, X_{t-1}, \ldots, X_{t-\infty}) \]
Convolutional neural networks are finite response systems, true or false

- True
- False

An input at time T affects the output of the convolutional layers of the network for all time, true or false

- True
- False
Convolutional neural networks are finite response systems, true or false

- True
- False

An input at time $T$ affects the output of the convolutional layers of the network for all time, true or false

- True
- False
Examples of infinite response systems

\[ Y_t = f(X_t, Y_{t-1}) \]

- Required: Define initial state: \( Y_{-1} \) for \( t = 0 \)
- An input at \( X_0 \) at \( t = 0 \) produces \( Y_0 \)
- \( Y_0 \) produces \( Y_1 \) which produces \( Y_2 \) and so on until \( Y_\infty \) even if \( X_1 \ldots X_\infty \) are 0
  - i.e. even if there are no further inputs!
- A single input influences the output for the rest of time!

- This is an instance of a NARX network
  - “nonlinear autoregressive network with exogenous inputs”
  - \( Y_t = f(X_{0:t}, Y_{0:t-1}) \)
- Output contains information about the entire past
A one-tap NARX network

- A NARX net with recursion from the output
A one-tap NARX network

- A NARX net with recursion from the output
A one-tap NARX network

- A NARX net with recursion from the output
A one-tap NARX network

- A NARX net with recursion from the output
A one-tap NARX network

- A NARX net with recursion from the output
• A NARX net with recursion from the output
A one-tap NARX network

- A NARX net with recursion from the output
A one-tap NARX network

- A NARX net with recursion from the output
A more complete representation

- A NARX net with recursion from the output
- Showing all computations
- All columns are identical
- An input at $t=0$ affects outputs forever
A NARX net with recursion from the output
Showing all computations
All columns are identical
An input at $t=0$ affects outputs forever

Brown boxes show output layers
All outgoing arrows are the same output
A more generic NARX network

- The output $Y_t$ at time $t$ is computed from the past $K$ outputs $Y_{t-1}, \ldots, Y_{t-K}$ and the current and past $L$ inputs $X_t, \ldots, X_{t-L}$
NARX Networks

• Very popular for time-series prediction
  – Weather
  – Stock markets
  – As alternate system models in tracking systems
    – Language

• Any phenomena with distinct “innovations” that “drive” an output

• Note: here the “memory” of the past is in the output itself, and not in the network
Let’s make memory more explicit

• Task is to “remember” the past
• Introduce an explicit memory variable whose job it is to remember

\[ m_t = r(y_{t-1}, h_{t-1}, m_{t-1}) \]
\[ h_t = f(x_t, m_t) \]
\[ y_t = g(h_t) \]

• \( m_t \) is a “memory” variable
  – Generally stored in a “memory” unit
  – Used to “remember” the past
- Memory unit simply retains a running average of past outputs
    - Input is constant (called a “plan”)
    - Objective is to train net to produce a specific output, given an input plan
  - Memory has fixed structure; does not “learn” to remember
    - The running average of outputs considers entire past, rather than immediate past
Elman Networks

- Separate memory state from output
  - “Context” units that carry historical state
    - For the purpose of training, this was approximated as a set of T independent 1-step history nets

- Only the weight from the memory unit to the hidden unit is learned
  - But during training no gradient is backpropagated over the “1” link
Training Elman Networks

- Separate memory state from output
  - “Context” units that carry historical state
    - For the purpose of training, this was approximated as a set of T independent 1-step history nets

- Only the weight from the memory unit to the hidden unit is learned
  - But during training no gradient is backpropagated over the “1” link
Story so far

• In time series analysis, models must look at past inputs along with current input
  – Looking at a finite horizon of past inputs gives us a convolutional network
• Looking into the infinite past requires recursion

• NARX networks recurse by feeding back the output to the input
  – May feed back a finite horizon of outputs

• “Simple” recurrent networks:
  – Jordon networks maintain a running average of outputs in a “memory” unit
  – Elman networks store hidden unit values for one time instant in a “context” unit
  – “Simple” (or partially recurrent) because during learning current error does not actually propagate to the past
    • “Blocked” at the memory units in Jordan networks
    • “Blocked” at the “context” unit in Elman networks
Poll 2 (@748, @749)

Memory neuron models have true recurrence, true or false

- True
- False

Memory neuron networks dedicate neurons specifically to store past history, true or false

- True
- False
Poll 2

Memory neuron models have true recurrence, true or false

- True
- False

Memory neuron networks dedicate neurons specifically to store past history, true or false

- True
- False
An alternate model for infinite response systems: the state-space model

\[ h_t = f(x_t, h_{t-1}) \]
\[ y_t = g(h_t) \]

- $h_t$ is the *state* of the network
  - *State* summarizes information about the entire past
    - Model directly embeds the memory in the state
- Need to define initial state $h_{-1}$
- This is a *fully recurrent* neural network
  - Or simply a *recurrent neural network*
The simple state-space model

- The state (green) at any time is determined by the input at that time, and the state at the previous time.
- An input at \( t=0 \) affects outputs forever.
- Also known as a recurrent neural net.

Initial value
An alternate model for infinite response systems: the state-space model

\[ h_t = f(x_t, h_{t-1}) \]
\[ y_t = g(h_t) \]

- \( h_t \) is the state of the network
- Need to define initial state \( h_{-1} \)
- The state can be arbitrarily complex
**Single hidden layer RNN**

- Recurrent neural network
- All columns are identical
- An input at \( t=0 \) affects outputs forever
Multiple recurrent layer RNN

- Recurrent neural network
- All columns are identical
- An input at $t=0$ affects outputs forever
Multiple recurrent layer RNN

- We can also have skips..
A more complex state

- All columns are identical
- An input at $t=0$ affects outputs forever
Or the network may be even more complicated

- Shades of NARX
- All columns are identical
- An input at $t=0$ affects outputs forever
Generalization with other recurrences

- All columns (including incoming edges) are identical
The simplest structures are most popular

- Recurrent neural network
- All columns are identical
- An input at $t=0$ affects outputs forever
A Recurrent Neural Network

- Simplified models often drawn
- The loops imply recurrence
The detailed version of the simplified representation

\[
\begin{align*}
X(t) \quad h(-1) \\
Y(t)
\end{align*}
\]
Multiple recurrent layer RNN

\[ Y(t) = f(h^{(2)}(-1), h^{(1)}(-1), X(t)) \]

- \( h^{(2)}(-1) \)
- \( h^{(1)}(-1) \)
- \( X(t) \)
- \( Y(t) \)

Time

\( t=0 \)
Multiple recurrent layer RNN
Equations

- Note superscript in indexing, which indicates layer of network from which inputs are obtained.
- Assuming vector function at output, e.g. softmax.
- The *state* node activation, $f_1()$ is typically $\tanh()$.
- Every neuron also has a *bias* input.

\[
h_i^{(1)}(t) = f_1 \left( \sum_j w_{ji}^{(1)} X_j(t) + \sum_j w_{ji}^{(11)} h_j^{(1)}(t - 1) + b_i^{(1)} \right)
\]

\[
Y(t) = f_2 \left( \sum_j w_{jk}^{(2)} h_j^{(1)}(t) + b_k^{(2)}, k = 1..M \right)
\]

$h_i^{(1)}(-1) = \text{part of network parameters}$
Equations

$h^{(1)}(-1) = \text{part of network parameters}$

- Computation:
  \[
  h^{(1)}(t) = f_1(W^{(1)}X(t) + W^{(11)}h^{(1)}(t - 1) + b^{(1)})
  \]
  \[
  Y(t) = f_2(W^{(2)}h^{(1)}(t) + b^{(2)})
  \]

- The recurrent state activation $f_1()$ is typically $\tanh()$
Equations

- Computation:
  \[
  h^{(1)}(t) = f_1\left( W^{(1)}X(t) + W^{(11)}h^{(1)}(t - 1) + b^{(1)} \right)
  \]
  \[
  Y(t) = f_2\left( W^{(2)}h^{(1)}(t) + b^{(2)} \right)
  \]

- The recurrent state activation \( f_1() \) is typically \( \tanh() \)
Equations

\[ h_i^{(1)}(-1) = \text{part of network parameters} \]
\[ h_i^{(2)}(-1) = \text{part of network parameters} \]

\[ h_i^{(1)}(t) = f_1 \left( \sum_j w_{ji}^{(1)} X_j(t) + \sum_j w_{ji}^{(11)} h_j^{(1)}(t - 1) + b_i^{(1)} \right) \]

\[ h_i^{(2)}(t) = f_2 \left( \sum_j w_{ji}^{(2)} h_j^{(1)}(t) + \sum_j w_{ji}^{(22)} h_j^{(2)}(t - 1) + b_i^{(2)} \right) \]

\[ Y(t) = f_3 \left( \sum_j w_{jk}^{(3)} h_j^{(2)}(t) + b_k^{(3)}, k = 1..M \right) \]

- Assuming vector function at output, e.g. softmax \( f_3() \)
- The *state* node activations, \( f_k() \) are typically \( \text{tanh}() \)
- Every neuron also has a *bias* input
\( h^{(1)}(-1) \) and \( h^{(2)}(-1) = \text{part of network parameters} \)

- Computation:

\[
\begin{align*}
    h^{(1)}(t) &= f_1 \left( W^{(1)} X(t) + W^{(11)} h^{(1)}(t - 1) + b^{(1)} \right) \\
    h^{(2)}(t) &= f_2 \left( W^{(2)} h^{(1)}(t) + W^{(22)} h^{(2)}(t - 1) + b^{(2)} \right) \\
    Y(t) &= f_3 \left( W^{(3)} h^{(2)}(t) + b^{(3)} \right)
\end{align*}
\]

- The recurrent state activation is typically \( \tanh() \)
Equations

\[ h^{(1)}(-1) = \text{part of network parameters} \]
\[ h^{(2)}(-1) = \text{part of network parameters} \]

\[ h^{(1)}(t) = f_1(W^{(01)}X(t) + W^{(11)}h^{(1)}(t - 1) + b^{(1)}) \]

\[ h^{(2)}(t) = f_2(W^{(12)}h^{(1)}(t) + W^{(02)}X(t) + W^{(22)}h^{(2)}(t - 1) + b^{(2)}) \]

\[ Y(t) = f_3(W^{(23)}h^{(2)}(t) + W^{(13)}h^{(1)}(t) + b^{(3)}) \]
Variants on recurrent nets

1: Conventional MLP
2: Sequence generation, e.g. image to caption
3: Sequence based prediction or classification, e.g. Speech recognition, text classification
Variants

1: *Delayed* sequence to sequence, e.g. machine translation
2: Sequence to sequence, e.g. stock problem, label prediction
Etc...

Images from Karpathy
Story so far

- Time series analysis must consider past inputs along with current input
- Looking into the infinite past requires recursion
- NARX networks achieve this by feeding back the output to the input
- “Simple” recurrent networks maintain separate “memory” or “context” units to retain some information about the past
  - But during learning the current error does not influence the past
- State-space models retain information about the past through recurrent hidden states
  - These are “fully recurrent” networks
  - The initial values of the hidden states are generally learnable parameters as well
- State-space models enable current error to update parameters in the past
How do we *train* the network

- Back propagation through time (BPTT)
- Given a collection of *sequence* inputs
  - \((X_i, D_i)\), where
  - \(X_i = X_{i,0}, \ldots, X_{i,T}\)
  - \(D_i = D_{i,0}, \ldots, D_{i,T}\)
- Train network parameters to minimize the error between the output of the network \(Y_i = Y_{i,0}, \ldots, Y_{i,T}\) and the desired outputs
  - This is the most generic setting. In other settings we just “remove” some of the input or output entries
Training the RNN

- The “unrolled” computation is just a giant shared-parameter neural network
  - All columns are identical and share parameters

- Network parameters can be trained via gradient-descent (or its variants) using shared-parameter gradient descent rules
  - Gradient computation requires a forward pass, back propagation, and pooling of gradients (for parameter sharing)
Training: Forward pass

- For each training input:
- Forward pass: pass the entire data sequence through the network, generate outputs
Recurrent Neural Net

Assuming time-synchronous output

# Assuming h(-1,*) is known
# Assuming L hidden-state layers and an output layer
# \( W_c(*) \) and \( W_r(*) \) are matrices, \( b(*) \) are vectors
# \( W_c \) are weights for inputs from current time
# \( W_r \) is recurrent weight applied to the previous time
# \( W_o \) are output layer weights

for \( t = 0:T-1 \)  # Including both ends of the index
    \( h(t,0) = x(t) \)  # Vectors. Initialize \( h(0) \) to input
for \( l = 1:L \)  # hidden layers operate at time \( t \)
    \( z(t,l) = W_c(l)h(t,l-1) + W_r(l)h(t-1,l) + b(l) \)
    \( h(t,l) = \tanh(z(t,l)) \)  # Assuming \( \tanh \) activ.
\( z_o(t) = W_o h(t,L) + b_o \)
\( Y(t) = \text{softmax}(z_o(t)) \)

Subscript “c” - current
Subscript “r” - recurrent
Training: Computing gradients

- For each training input:
  - Backward pass: Compute gradients via backpropagation
    - Back Propagation Through Time
Back Propagation Through Time

Will only focus on one training instance

All subscripts represent components and not training instance index
Back Propagation Through Time

DIV

The divergence computed is between the sequence of outputs by the network and the desired sequence of outputs.
- DIV is a scalar function of a series of vectors!

- This is not just the sum of the divergences at individual times
  - Unless we explicitly define it that way
• \( Y(t) \) is the output at time \( t \)
  – \( Y_i(t) \) is the \( i \)th output
• \( Z^{(2)}(t) \) is the pre-activation value of the neurons at the output layer at time \( t \)
• \( h(t) \) is the output of the hidden layer at time \( t \)
  – Assuming only one hidden layer in this example
• \( Z^{(1)}(t) \) is the pre-activation value of the hidden layer at time \( t \)
Notation

- \( W^{(1)} = \begin{bmatrix} w_{ij}^{(1)} \end{bmatrix} \) is the matrix of current weights from the input to the hidden layer.
- \( W^{(2)} = \begin{bmatrix} w_{ij}^{(2)} \end{bmatrix} \) is the matrix of current weights from the hidden layer to the output layer.
- \( W^{(11)} = \begin{bmatrix} w_{ij}^{(11)} \end{bmatrix} \) is the matrix of recurrent weights from the hidden layer to itself.
Back Propagation Through Time

First step of backprop: Compute $\nabla_{Y(T)}DIV$ (Compute $\frac{dDIV}{dY_i(T)}$ for all $i$)

Note: DIV is a function of all outputs $Y(0)$ ... $Y(T)$

In general we will be required to compute $\frac{dDIV}{dY_i(t)}$ for all $i$ and $t$ as we will see. This can be a source of significant difficulty in many scenarios.
Rules we will use

\[ \nabla_{Z_i} L = \nabla_{Y_i} L \nabla_{Z_i} Y_i \]

\[ \nabla_{Y_{i-1}} L = \nabla_{Z_i} L \ W \]

\[ \nabla_{W} L = Y_{i-1} \nabla_{Z_i} L \]

\[ \nabla_{Y_i} L = \nabla_{Z_i} L \ W_1 + \nabla_{Z_{i+1}} L \ W_2 \]
Special case, when the overall divergence is a simple sum of local divergences at each time: $DIV = \sum_t Div(t)$

Must compute $\nabla_{Y(t)} DIV$

Will get $\nabla_{Y(t)} Div(t)$

$$\frac{\partial DIV}{\partial Y_i(t)} \text{ for all } i \text{ for all } T$$

$$\frac{\partial DIV}{\partial Y_i(t)} = \frac{\partial Div(t)}{\partial Y_i(t)}$$
Back Propagation Through Time

First step of backprop: Compute $\frac{dDIV}{dY_i(T)}$ for all $i$

$$\nabla_{Z^{(2)}(T)} DIV = \nabla_{Y(T)} DIV \nabla_{Z^{(2)}(T)} Y(T)$$

$$\frac{dDIV}{dZ_i^{(2)}(T)} = \frac{dDIV}{dY_i(T)} \frac{dY_i(T)}{dZ_i^{(2)}(T)}$$

OR

$$\frac{dDIV}{dZ_i^{(2)}(T)} = \sum_j \frac{dDIV}{dY_j(T)} \frac{dY_j(T)}{dZ_i^{(2)}(T)}$$
Back Propagation Through Time

\[ \frac{d\text{DIV}}{dh_i(T)} \text{ for all } i \]

\[ \frac{d\text{DIV}}{dZ_i^{(2)}(T)} = \frac{d\text{Div}(T)}{dY_i(T)} \frac{dY_i(T)}{dZ_i^{(2)}(T)} \]

\[ \nabla_{h(T)} \text{DIV} = \nabla_{Z^{(2)}(T)} \text{DIV} W^{(2)} \]
Back Propagation Through Time

$$h_{-1}$$

$$Y(0)$$  $$Y(1)$$  $$Y(2)$$  $$Y(T-2)$$  $$Y(T-1)$$  $$Y(T)$$

$$X(0)$$  $$X(1)$$  $$X(2)$$  $$X(T-2)$$  $$X(T-1)$$  $$X(T)$$

$$D(1..T)$$

$$\frac{dDIV}{dZ_i^{(2)}(T)} = \frac{dDiv(T)}{dY_i(T)} \frac{dY_i(T)}{dZ_i^{(2)}(T)}$$

$$\frac{dDIV}{dh_i(T)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dZ_j^{(2)}(T)}$$

$$\nabla_W^{(2)}DIV = h(T)\nabla_{Z_i^{(2)}(T)}DIV$$

$$\frac{dDIV}{dw_{ij}^{(2)}} = \frac{dDIV}{dZ_j^{(2)}(T)} h_i(T)$$
Back Propagation Through Time

\[ \nabla_{Z^{(1)}(T)} DIV = \nabla_{h(T)} DIV \nabla_{Z^{(1)}(T)} h(T) \]

\[ \frac{dDIV}{dZ_{i}^{(2)}(T)} = \frac{dDIV}{dY_{i}(T)} \frac{dY_{i}(T)}{dZ_{i}^{(2)}(T)} \]

\[ \frac{dDIV}{dh_{i}(T)} = \sum_{j} w_{ij}^{(2)} \frac{dDIV}{dZ_{j}^{(2)}(T)} \]

\[ \frac{dDIV}{dw_{ij}^{(2)}} = \frac{dDIV}{dz_{j}^{(2)}(T)} h_{i}(T) \]
Back Propagation Through Time

\[ \nabla_{W^{(1)}} DIV = X(T) \nabla_{Z^{(1)}(T)} DIV \]

\[ \frac{dDIV}{dw_{ij}^{(1)}} = \frac{dDIV}{dZ_{j}^{(1)}(T')} X_{i}(T) \]
Back Propagation Through Time

\[ dDIV = \frac{dDIV}{dw_{ij}^{(1)}} = \frac{dDIV}{dZ_j^{(1)}(T)} X_i(T) \]

\[ dDIV = \frac{dDIV}{dw_{ij}^{(11)}} = \frac{dDIV}{dZ_j^{(1)}(T)} h_i(T - 1) \]

\[ \nabla_{W^{(11)}} DIV = h(T - 1) \nabla_{Z^{(1)}(T)} DIV \]
Back Propagation Through Time

\( \nabla^{(2)}_{Z(T-1)} DIV = \nabla^{(2)}_{Y(T-1)} DIV \nabla^{(2)}_{Z(T-1)} Y(T-1) \)

\[
\frac{dDIV}{dZ_i^{(2)}(T-1)} = \frac{dDIV}{dY_i(T-1)} \frac{dY_i(T-1)}{dZ_i^{(2)}(T-1)}
\]

OR

\[
\frac{dDIV}{dZ_i^{(2)}(T-1)} = \sum_j \frac{dDIV}{dY_j(T-1)} \frac{dY_j(T-1)}{dZ_i^{(2)}(T-1)}
\]
Back Propagation Through Time

\[ \frac{dDIV}{dh_i(T - 1)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dZ_j^{(2)}(T - 1)} + \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(1)}(T)} \]

\[ \nabla_{h(T-1)} DIV = \nabla_{Z^{(2)(T-1)}} DIV W^{(2)} + \nabla_{Z^{(1)(T)}} DIV W^{(11)} \]
Back Propagation Through Time

\[
\frac{dDIV}{dh_i(T - 1)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dZ_j^{(2)}(T - 1)} + \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(1)}(T)}
\]

Note the addition

\[
\frac{dDIV}{dw_{ij}^{(2)}} + = \frac{dDIV}{dZ_j^{(2)}(T - 1)} h_i(T - 1)
\]

Note the addition

\[
\nabla_{W^{(2)}} DIV + = h(T - 1)\nabla_{Z^{(2)}(T-1)} DIV
\]
Back Propagation Through Time

\[
\begin{align*}
\frac{d\text{DIV}}{dZ_i^{(1)}(T - 1)} &= \frac{d\text{DIV}}{dh_i(T - 1)} \frac{dh_i(T - 1)}{dZ_i^{(1)}(T - 1)} \\
\nabla_{Z^{(1)}_i(T-1)} \text{DIV} &= \nabla_{h(T-1)} \text{DIV} \nabla_{Z^{(1)}_i(T-1)} h(T - 1)
\end{align*}
\]
Back Propagation Through Time

**DIV**

\[
\frac{dDIV}{dZ_i^{(1)}(T-1)} = \frac{dDIV}{dh_i(T-1)} \frac{dh_i(T-1)}{dZ_i^{(1)}(T-1)}
\]

\[
\nabla_W^{(1)} DIV = X(T-1) \nabla_{Z^{(1)}(T-1)} DIV
\]

Note the addition
Back Propagation Through Time

Note the addition

Note the addition
Back Propagation Through Time

Continue computing derivatives going backward through time until...

\[
\frac{dDIV}{dh_i(-1)} = \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(1)}(0)}
\]

\[
\nabla_{h_{-1}} DIV = \nabla_{Z^{(1)}(0)} DIV W^{(11)}
\]
Back Propagation Through Time

For $t = T$ downto 0

\[ \nabla_{Z^{(2)}(t)} DIV = \nabla_{Y(t)} DIV \nabla_{Z^{(2)}(t)} Y(t) \]
\[ \nabla_{h(t)} DIV = \nabla_{Z^{(2)}(t)} DIV W^{(2)} + \nabla_{Z^{(1)}(t+1)} DIV W^{(11)} \]
\[ \nabla_{Z^{(1)}(t)} DIV = \nabla_{h(t)} DIV \nabla_{Z^{(1)}(t)} h(t) \]

\[ \nabla_{W^{(2)}} DIV += h(t) \nabla_{Z^{(2)}(t)} DIV \]
\[ \nabla_{W^{(11)}} DIV += h(t-1) \nabla_{Z^{(1)}(t+1)} DIV \]
\[ \nabla_{W^{(1)}} DIV += X(t) \nabla_{Z^{(1)}(t)} DIV \]
\[ \nabla_{h_{-1}} DIV = \nabla_{Z^{(1)}(0)} DIV W^{(11)} \]
Back Propagation Through Time

\[
\frac{dDIV}{dh_i^{(k)}(t)} = \sum_j w_{i,j}^{(k+1)} \frac{dDIV}{dZ_j^{(k+1)}(t)} + \sum_j w_{i,j}^{(k,k)} \frac{dDIV}{dZ_j^{(k)}(t + 1)}
\]

Not showing derivatives at output neurons

\[
\frac{dDIV}{dZ_i^{(k)}(t)} = \frac{dDIV}{dh_i^{(k)}(t)} f_k'(Z_i^{(k)}(t))
\]
Back Propagation Through Time

\[
\frac{dDIV}{dh_i(-1)} = \sum_j w_{ij}^{(11)} \frac{dDIV}{dZ_j^{(1)}(0)}
\]

\[
\frac{dDIV}{dw_{ij}^{(1)}} = \sum_t \frac{dDIV}{dZ_j^{(1)}(t)} X_i(t)
\]

\[
\frac{dDIV}{dw_{ij}^{(11)}} = \sum_t \frac{dDIV}{dZ_j^{(1)}(t)} h_i(t-1)
\]
# Assuming forward pass has been completed
# Jacobian(x,y) is the jacobian of x w.r.t. y
# Assuming dY(t) = gradient(div,Y(t)) available for all t
# Assuming all dz, dh, dW and db are initialized to 0

for t = T-1:downto:0  # Backward through time
    dz_o(t) = dY(t)Jacobian(Y(t),z_o(t))
    dW_o += h(t,L)dz_o(t)
    db_o += dz_o(t)
    dh(t,L) += dz_o(t)W_o

for l = L:1  # Reverse through layers
    dz(t,l) = dh(t,l)Jacobian(h(t,l),z(t,l))
    dh(t,l-1) += dz(t,l) W_c(l)
    dh(t-1,l) = dz(t,l) W_r(l)

    dW_c(l) += h(t,l-1)dz(t,l)
    dW_r(l) += h(t-1,l)dz(t,l)
    db(l) += dz(t,l)
• Can be generalized to any architecture
SGD trains neural networks one input at a time, rather than over batches of inputs. The corresponding equivalent for RNNs would be to update the network after each input vector: True or False

- True
- False

Select all that are true:

- In RNNs the divergence we minimize is the sum of the divergences for the individual inputs in the time series
- The divergence is the divergence between the actual sequence of outputs and the desired sequence of outputs and cannot always be decomposed into the sum of divergences at individual time steps.
SGD trains neural networks one input at a time, rather than over batches of inputs. The corresponding equivalent for RNNs would be to update the network after each input vector: True or False

- True
- False

Select all that are true:

- In RNNs the divergence we minimize is the sum of the divergences for the individual inputs in the time series
- The divergence is the divergence between the actual sequence of outputs and the desired sequence of outputs and cannot always be decomposed into the sum of divergences at individual time steps.
Extensions to the RNN: *Bidirectional RNN*

- In problems where the entire input sequence is available before we compute the output, RNNs can be bidirectional

- RNN with both forward and backward recursion
  - Explicitly models the fact that just as the future can be predicted from the past, the past can be deduced from the future

Proposed by Schuster and Paliwal 1997

*Bidirectional RNN (BRNN)*

Alex Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*
• “Block” performs bidirectional inference on input
  – “Input” could be input series X(0)...X(T) or the output of a previous layer (or block)

• The Block has two components
  – A forward net process the data from t=0 to t=T
  – A backward net processes it backward from t=T down to t=0
Bidirectional RNN block

- The forward net process the data from $t=0$ to $t=T$
  - Only computing the hidden state values.
The backward nets processes the input data in reverse time, end to beginning:

- Initially only the hidden state values are computed:
  - Clearly, this is not an online process and requires the entire input data.
  - Note: This is not the backward pass of backprop.
The computed states of both networks are combined to give you the output of the bidirectional block.

- Typically just concatenate them

\[ h(t) = [h_f(t); h_b(t)] \]
• Actual network may be formed by stacking many independent bidirectional blocks followed by an output layer
  — Forward and backward nets in each block are a single layer
• Or by a single bidirectional block followed by an output layer
  — The forward and backward nets may have several layers
• In either case, it’s sufficient to understand forward inference and backprop rules for a single block
  — Full forward or backprop computation simply requires repeated application of these rules
A day trader in the stock exchange uses RNNs to make predictions from historical market data to decide which stocks to buy. He can use a bidirectional RNN for his task.

- True
- False
A day trader in the stock exchange uses RNNs to make predictions from historical market data to decide which stocks to buy. He can use a bidirectional RNN for his task.

- True
- False
Bidirectional RNN block: inference

# Subscript f represents forward net, b is backward net
# Assuming h_f(-1,*) and h_b(inf,*) are known
# x(t) is the input to the block (which could be from a lower layer)

#forward recurrence
for t = 0:T-1  # Going forward in time
    h_f(t,0) = x(t)  # Vectors. Initialize h_f(0) to input
    for l = 1:L_f  # L_f is depth of forward network hidden layers
        z_f(t,l) = W_{fc}(l)h_f(t,l-1) + W_{fr}(l)h_f(t-1,l) + b_f(l)
        h_f(t,l) = tanh(z_f(t,l))  # Assuming tanh activ.

#backward recurrence
h_b(T,:, :) = h_b(inf,:, :)  # Just the initial value
for t = T-1:downto:0  # Going backward in time
    h_b(t,0) = x(t)  # Vectors. Initialize h_b(0) to input
    for l = 1:L_b  # L_b is depth of backward network hidden layers
        z_b(t,l) = W_{bc}(l)h_b(t,l-1) + W_{br}(l)h_b(t+1,l) + b_b(l)
        h_b(t,l) = tanh(z_b(t,l))  # Assuming tanh activ.

for t = 0:T-1  # The output combines forward and backward
    h(t) = [h_f(t,L_f); h_b(t,L_b)]
Bidirectional RNN: Simplified code

• Code can be made modular and simplified for better interpretability...
First: Define forward recurrence

# Inputs:
#    L : Number of hidden layers
#    Wc,Wr,b: current weights, recurrent weights, biases
#    hinit: initial value of h(representing h(-1,*))
#    x: input vector sequence
#    T: Length of input vector sequence
# Output:
#    h, z: sequence of pre-and post activation hidden
#          representations from all layers of the RNN

function RNN_forward(L, Wc, Wr, b, hinit, x, T)
    h(-1,:) = hinit # hinit is the initial value for all layers
    for t = 0:T-1 # Going forward in time
        h(t,0) = x(t) # Vectors. Initialize h(0) to input
        for l = 1:L
            z(t,l) = Wc(l)h(t,l-1) + Wr(l)h(t-1,l) + b(l)
            h(t,l) = tanh(z(t,l)) # Assuming tanh activ.
        end
    end
    return h
Bidirectional RNN block

# Subscript f represents forward net, b is backward net
# Assuming $h_f(-1,*)$ and $h_b(inf,*)$ are known

#forward pass
$h_f = \text{RNN}_\text{forward}(L_f, W_{fc}, W_{fr}, b_f, h_f(-1,:), x, T)$

#backward pass
$x_{rev} = \text{fliplr}(x)$  # Flip it in time
$h_{brev} = \text{RNN}_\text{forward}(L_b, W_{bc}, W_{br}, b_b, h_b(inf,:), x_{rev}, T)$
$h_b = \text{fliplr}(h_{brev})$  # Flip back to straighten time

#combine the two for the output
for $t = 0:T-1$  # The output combines forward and backward
$h(t) = [h_f(t,L_f); h_b(t,L_b)]$
• **Forward pass:** Compute both forward and backward networks and final output
Backpropagation in BRNNs

- Backward pass: Assume gradients of the divergence are available for the block outputs $h(t)$
  - Obtained via backpropagation from network output
  - Will have the same dimension (length) as $h(t)$
    - Which is the sum of the dimensions of $h_f(t)$ and $h_b(t)$
Backpropagation in BRNNs

- Separate gradient into forward and backward components
  \[ \nabla_{h(t)} Div = [\nabla_{h_f(t)} Div; \nabla_{h_b(t)} Div] \]
  - Extract \( \nabla_{h_f(t)} Div \) and \( \nabla_{h_b(t)} Div \) from \( \nabla_{h(t)} Div \).
- Separately perform backprop on the forward and backward nets
• Backprop for forward net:
  – Backpropagate $\nabla_{h_f(t)} Div$ from $t = T$ down to $t = 0$ in the usual way
  – Will obtain derivatives for all the parameters of the forward net
  – Will also get $\nabla_{X(t)} Div_{forward}$
    • Partial derivative of the gradient for $X(t)$ computed through the forward net
Backpropagation in BRNNs

- Backprop for backward net:
  - Backpropagate $\nabla_{h_b(t)} Div$ forward from $t = 0$ up to $t = T$
  - Will obtain derivatives for all the parameters of the forward net
  - Will also get $\nabla_{X(t)} Div_{backward}$
    - Partial derivative of the gradient for $X(t)$ computed through the backward net
Backpropagation in BRNNs

- Finally add up the forward and backward partial derivatives to get the full gradient for $X(t)$

$$\nabla_{X(t)} Div = \nabla_{X(t)} Div_{backward} + \nabla_{X(t)} Div_{forward}$$
Backpropagation: Pseudocode

- As before we will use a 2-step code:
  - A basic backprop routine that we will call
  - Two calls to the routine within a higher-level wrapper
First: backprop through a recurrent net

# Inputs:
# (In addition to inputs used by L : Number of hidden layers
# dh_top: derivatives ddiv/dh_(t,L) at each time (*) may be f or b
# h, z: h and z values returned by the forward pass
# T: Length of input vector sequence
# Output:
# dW^c, dW^r, db dh_{init}: derivatives w.r.t current and recurrent weights,
# biases, and initial h.
# Assuming all dz, dh, dW^c, dW^r and db are initialized to 0

function RNN_bptt(L, W^c, W^r, b, hinit, x, T, dh_top, h, z)
    dh = zeros
    for t = T-1:downto:0  # Backward through time
        dh(t,L) += dh_top(t)
        h(t,0) = x(t)
    for l = L:1  # Reverse through layers
        dz(t,l) = dh(t,l)Jacobian(h(t,l),z(t,l))
        dh(t,l-1) += dz(t,l) W^c(l)
        dh(t-1,l) += dz(t,l) W^r(l)
        dW^c(l) += h(t,l-1)dz(t,l)
        dW^r(l) += h(t-1,l)dz(t,l)
        db(l) += dz(t,l)
        dx(t)= dh(t,0)
    return dx, dW^c, dW^r, db, dh(-1)  # dh(-1) is actually dh(-1,1:L,:)
BRNN block: gradient computation

# Subscript f represents forward net, b is backward net
# Given dh(t), t=0...T-1 : The sequence of gradients from the upper layer
# Also assumed available:
#     x(t), t=0...T-1 : the input to the BRNN block
#     z_f(t), h_f(t) : Complete forward-computation outputs for all layers of the forward net
#     z_b(t), h_b(t) : Complete backward-computation outputs for all layers of the backward net
# L_f and L_b are the number of components in h_f(t) and h_b(t)

for t = 0:T-1  # Separate out forward and backward net gradients
    dh_f(t) = dh(t,1:L_f)
    dh_b(t) = dh(t,L_f+1:L_f+L_b)

#forward net
[dx_f dW_{fc}, dW_{fr}, db_f, dh_f(-1)] = RNN_bptt(L, W_{fc}, W_{fr}, b_f, h_f(-1), x, T, dh_f, h_f, z_f)

#backward net
x_rev = fliplr(x)  # Flip it in time
dh_brev = fliplr(dh_b)
h_brev = fliplr(h_b)
z_brev = fliplr(z_b)
[dx_brev, dW_{bc}, dW_{br}, db_b, dh_b(\inf)] = RNN_bptt(L, W_{bc}, W_{br}, b_b, h_b(\inf), x_rev, T, dh_brev, h_brev, z_brev)
dx_b = fliplr(dx_brev)

for t = 0:T-1  # Add the partials
    dx(t) = dx_f(t) + dx_b(t)
Story so far

• Time series analysis must consider past inputs along with current input

• Recurrent networks look into the infinite past through a state-space framework
  – Hidden states that recurse on themselves

• Training recurrent networks requires
  – Defining a divergence between the actual and desired output sequences
  – Backpropagating gradients over the entire chain of recursion
    • Backpropagation through time
  – Pooling gradients with respect to individual parameters over time

• Bidirectional networks analyze data both ways, begin→end and end→beginning to make predictions
  – In these networks, backprop must follow the chain of recursion (and gradient pooling) separately in the forward and reverse nets
RNNs

• Excellent models for series data analysis tasks
  – Time-series prediction
  – Time-series classification
  – Sequence generation..