MODELS FOR SEQUENTIAL DATA
THE MARKOV ASSUMPTION

• Given the current (vector of) inputs, the next output is independent of the previous outputs.
  \[ P(y_t|x_t, x_{t-1}, \ldots, x_0) = P(y_t|x_t). \]

• Example: Basketball Prediction (open in Chrome)

• \textit{k}-order Markov process: next observation depends on fixed-length part of previous history.
  
  ➢ sliding window model
  
  ➢ convolutional neural net
MARKOV CHAIN MODEL

- At each point, the system is in a state $s_t$
- Given the state, the next output is independent of observations
  \[ P(y_t|s_t, x_t, x_{t-1}, ..., x_0) = P(y_t|s_t) \]
- Current state depends only on current observation and previous state
  \[ P(s_t|s_{t-1}, x_t, x_{t-1}, ..., x_0) = P(s_t|s_{t-1}, x_t) \]
A Markov chain model where the state is not observed. Like a cluster.

1. The output at time $t$ is independent of previous inputs given the (right) hidden state at time $t$
   \[ P(y_t|h_t, x_t, x_{t-1}, ..., x_0) = P(y_t|h_t). \]

2. The hidden state at time $t$ is independent of previous inputs given the previous hidden state at time $t-1$
   \[ P(h_t|h_{t-1}, x_t, x_{t-1}, ..., x_0) = P(h_t|h_{t-1}) \]
   • Number of hidden states $k$ is specified in advance.
   • Hidden state $\approx$ cluster of history.
RECURRENT NEURAL NET (RNN)

- Basic Idea: Like HMM where Hidden State $\rightarrow$ Activation Vector of hidden nodes.
- $P(y_t|h_t, x_t, x_{t-1}, ..., x_0) = P(y_t|h_t)$ where $h_t$ is the activation vector of hidden states
  
  $P(h_t|h_{t-1}, x_t, x_{t-1}, ..., x_0) = P(h_t|h_{t-1})$

- In terms of NN activations:
  
  $$h^i(t) = g_i \left( \sum_m w_{im} h^m(t-1) + w_{ii} h^i(t-1) \right)$$

  Hidden unit activation depends on its own previous activation

- If output = next observation, can use to generate sequences ($y_t = x_{t+1}$)

- Generation Demo
UNROLLED RNN

Rnn examples
LSTMS AND PROGRAMS
RNNS AND LONG-RANGE DEPENDENCIES

- Problem: The hidden units have to remember information. But when does past information become relevant to present prediction?
- Example: “In Korea, more than half of all the residents speak Korean”.
- RNNs have trouble learning long-range dependencies
- More technically, we have the vanishing/exploding gradient problem in unrolling.
  - Roughly, temporal chain rule $\Rightarrow$ product of gradients
    - gradients $< 1 \Rightarrow$ product close to 0
    - gradients $> 1 \Rightarrow$ product explodes
LONG SHORT-TERM MEMORY (LSTM)

- Motivation: improve ability to learn long-range dependencies.
- Complicated Model, intuitions:
  - Introduce special hidden units called “memory cells”.
  - Content of memory cells is carried from past to future on a “special track”.
  - More precisely: the current content of a memory cell has linear dependence on its previous content → gradients neither vanish nor explode.
  - What should be put into a memory cell? – “Input gates” learn to fill them.
  - When is the content of a memory cell relevant? – “Output gates” learn when to use it.
  - What if the content in our precious memory cells is no longer relevant? – “Forget gates” learn when to erase them.
LSTM CONNECTION DIAGRAM

a. Observation

hidden

memory cells

hidden

A a1

A a2

output

b. Cell input

memory cell

output gate

CEC

forget gate

input gate

cell output
• One way to think of a memory cell is that it is a probabilistic version of a variable in a traditional program.

• For a traditional program variable, you can assign it new values, retrieve the value when needed, update the value.

• A := 5
  begin
     ....
  end
  if A > 4 then output ...
PROCESSING KOREAN EXAMPLE

1. cell := empty /* initialize memory cell */
2. while not end_of_sequence
   1. current_word := read_next_word
   2. previous_cell := cell
   3. if input context is right
      cell := current_word /*e.g. after “In” store “Korea”
   else
      cell := previous_cell /*copy previous value*/
   4. if output context is right
      use cell to predict /* e.g. after “speak” predict “Korean” */

“In Korea, more than half of all the residents speak Korean”.
LSTM PROGRAM WITH DETERMINISTIC GATES

1. cell := empty /* initialize memory cell */
2. while not end_of_sequence
   1. current_word := read_next_word
   2. previous_cell := cell
   3. compute input_gate, forget_gate, output_gate activations using previous hidden node activations
      /* like an RNN */
   4. compute candidate_memory using previous hidden node activations
      /* without using previous_cell */
      /* like an RNN */
   5. if input_gate is on
      cell := candidate_memory
      elseif forget_gate is on
      cell := empty
      else cell := previous_cell
   6. if output_gate is on, predict using current cell activation, current hidden node activations
      /* like an RNN */
Figure 4.9: The architecture of LSTMs

The problem with standard RNNs is that while the goal is to remember things from far back, in practice they seem to forget quickly. In Figure 4.9 everything in the dotted box corresponds to a single RNN unit. Obviously LSTMs elaborate the architecture quite significantly. First, note that we have shown one copy of an LSTM in a back-prop-though-time diagram. So on the left we have information coming in from processing the previous word (using two tensors of information rather than one). At the bottom we have the next word coming in. On the right we have two tensors going out to inform the next time unit and, as in plain RNNs, we have this information going “up” in the diagram to predict the next word and the loss (upper right-hand side).

The goal is to improve the RNN’s memory of past events by training it to remember the important stuff and forget the rest. To this end, LSTMs pass two versions of the past. The “official” selective memory is at the top and a more local version at the bottom. The top memory timeline is called the cell state and abbreviated c. The lower line is called h.

Figure 4.9 introduces several new connectives and activation functions. First, we see that the memory line is modified at two locations before being passed on to the next time unit. They are labeled times (X), and plus (+). The idea is that memories are removed at the times unit, and added at the plus unit.
• Recall the RNN recurrence \( h^i(t) = g_i(\sum w_{im} h^m(t-1) + w_{ii} h^i(t-1)) \)

• Applies to hidden units and input/output/forget gates (ignoring current input \( x_i \) for now).

• The new memory cell contents is
  
  \[ s^c(t) = [h^{input^c}(t) \times g_c(\sum_m w_{cm} h^m(t-1))] + [h^{forget^c}(t) \times s^c(t-1)] \]

• The new memory cell output is the output gate \( \times \) cell activation

\[ h^c(t) = h^{out^c}(t) \times g_c(s^c(t)) \]
Several variations on the LSTM gates have been developed.
Especially popular is GRU
Intuition: replace “input” and “forget” by “update”
GRU EQUATIONS

• New content $h'_t = \tanh(Wx_t + r_t \odot Uh_{t-1})$

  Transform current observation $x_t$

  Transform previous hidden state $h_{t-1}$

  Reset gate for previous hidden state $r_t$

• Update hidden state $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$

  With probability $z_t$ keep previous hidden state

  With probability $1 - z_t$ adopt new content

• Reset gates $r_t$ and update gates $z_t$ are trainable function of previous hidden state and current input
CONCLUSION

- Hidden state at time $t = \text{summary of data up to time } t$
- Hidden layer in NN = distributed continuous representation of hidden state
- Recurrent NN: hidden state at time $t-1$ feeds into hidden state at time $t$
- Powerful method for learning with sequential data
- Problem: long-range dependencies and the vanishing gradient problem
- Possible solutions: gating with essentially linear updates (LSTMs, GRU)