Sequence modeling

Professor Marie Roch

San Diego State University

Sequence modeling

Basic ideas

- Sequences of vectors, e.g. frames in an audio stream
- Learn about subsequences

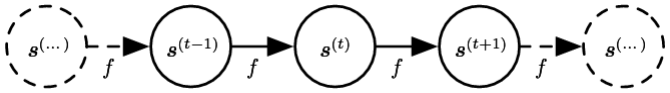
San Diego State University

Unfolding computational graphs

- Consider a function that relies on previous state: $s^{(t)} = f(s^{(t-1)}|\theta)$
- Next step:

$$s^{(t+1)} = f(s^{(t)}|\theta)$$
$$s^{(t+1)} = f(f(s^{(t-1)}|\theta)|\theta)$$

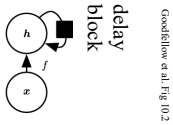
- We can think of this as a graph



Goodfellow et al. Fig 10.1

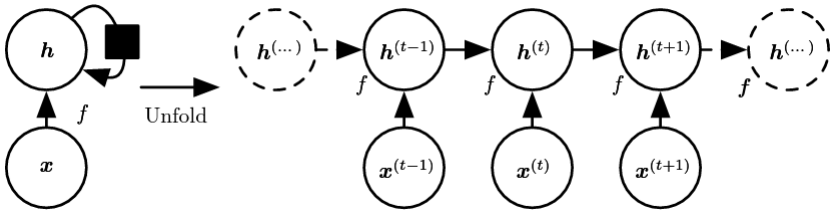
Add an input

- $s^{(t)} = f(s^{(t-1)}, x^{(t)}|\theta)$
- $s^{(t)}$ has past history of input
- Consider this occurring in a hidden unit: $h^{(t)}=f(h^{(t-1)}, x^{(t)}|\theta)$



Goodfellow et al. Fig 10.2

Recurrent neural networks (RNNs)



Goodfellow et al. Fig 10.2

- Prior knowledge of $x^{(t)}$ tends to be lossy, as one usually does not need the entire history

RNNs

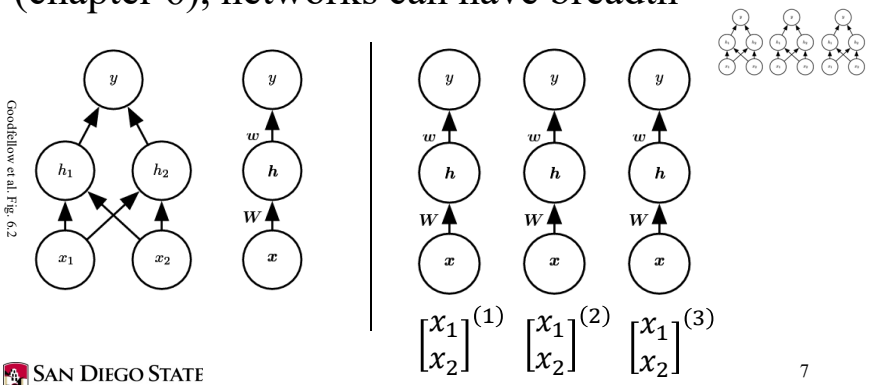
RNNs take a sequence and output:
- a sequence
- or a single response

In addition to the current input, nodes can depend on:
- outputs of previous nodes
- previous outputs

Recurrence in the hidden layers can result in Turing complete networks.

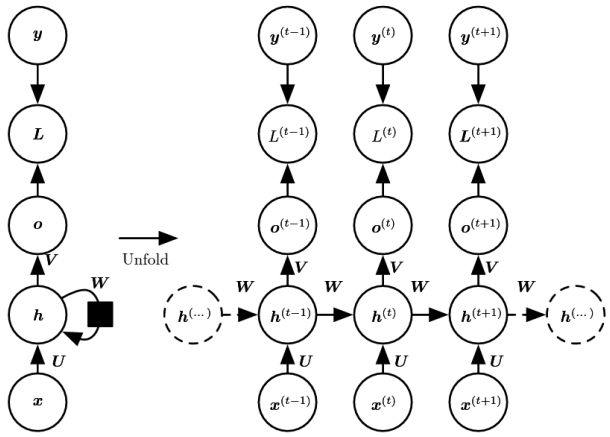
Notation

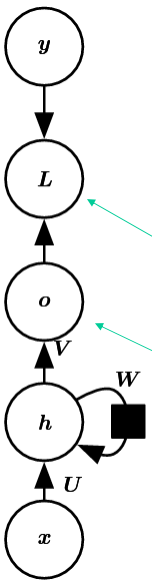
We are using compact network representation (chapter 6), networks can have breadth



Case 1 example

one output/input, depends on previous inputs



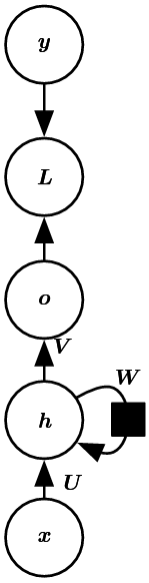


Case 1 example contd.

discrete (e.g. categorical) outputs

Loss(y,o) – Needed for training

output, e.g. unnormalized
log P(category|history)
For classification, can add a softmax layer for normalized probabilities, unnecessary for training



Case 1 example contd.

Forward propagation

Initialize state $h^{(0)}$

For each time t

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

$$h^{(t)} = \tanh\left(a^{(t)}\right)$$

$$o^{(t)} = c + Vh^{(t)}$$

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)})$$ (postprocessing step)

Loss of a sequence

- Sequence loss is sum of losses

$$L(\{x^{(1)}, x^{(2)}, \ldots, x^{(\tau)}\}, \{y^{(1)}, y^{(2)}, \ldots, y^{(\tau)}\}) = \sum_t L^{(t)}$$

- Suppose we use a negative log likelihood loss (e.g. MLE approach), this becomes

$$\sum_t L^{(t)} = -\sum_t \log P_{model}(y^{(t)} \mid x^{(1)}, x^{(2)}, \ldots, x^{(t)})$$

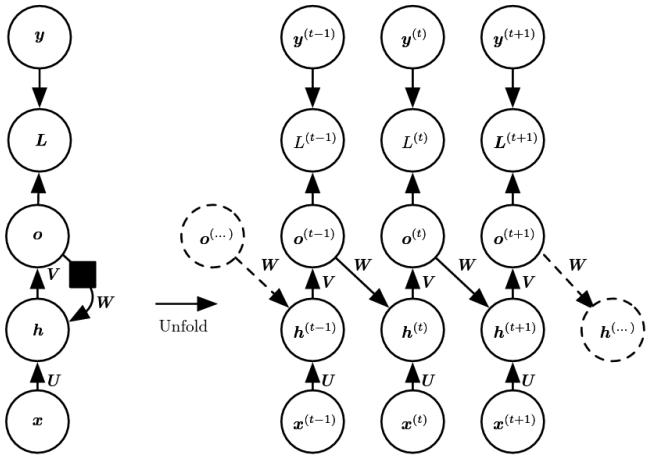
Gradient of sequence

(overview)

- Forward pass – Compute gradient of each step
- Backward pass – Use stored states and gradients to *back-propagate through time*
- Expensive in time and space

Case 2

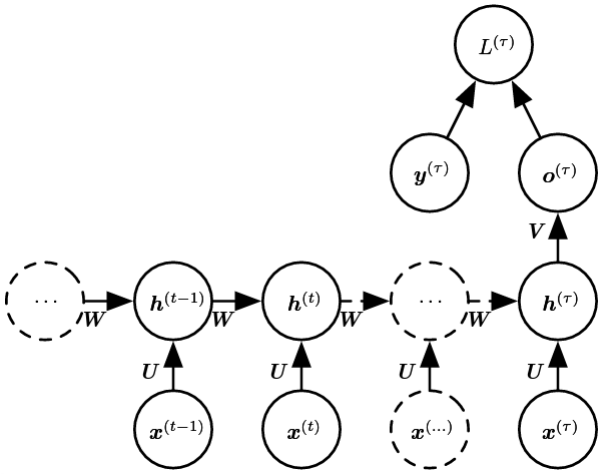
output depends on previous output



Goodfellow et al. Fig. 10.4

Case 3

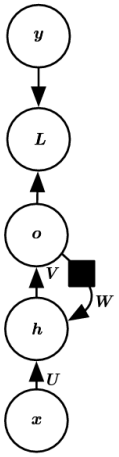
labeling a sequence



Goodfellow et al. Fig. 10.5

More on output recurrences

- Dependence on solely the output is not Turing complete
- Loss function minimizes difference between target and output; nothing will drive it to capture information about network state



Output recurrences: Teacher forcing

When training for output recurrences, we could use a MLE approach to find the model which maximizes:

$$\log P(o^{(1)}, y^{(2)} \mid x^{(1)}, x^{(2)})$$

but we know what we want for $o^{(1)}$: $y^{(1)}$

Teacher forcing

If we use the previous label

$$\log P(y^{(1)}, y^{(2)} \mid x^{(1)}, x^{(2)})$$

we can parallelize the backpropagation as we don't need to compute the previous dependency.

Teacher forcing

We can rewrite the MLE target (Bayes rule):

$$\log P(y^{(1)}, y^{(2)} \mid x^{(1)}, x^{(2)}) = \log\Big(P(y^{(2)} \mid y^{(1)}, x^{(1)}, x^{(2)}) P(y^{(1)} \mid x^{(1)}, x^{(2)})\Big)$$

$$= \log\Big(P(y^{(2)} \mid y^{(1)}, x^{(2)}) P(y^{(1)} \mid x^{(1)})\Big)$$

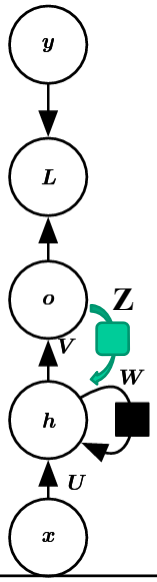
$P(y^{(t)} \mid x^{(1)}, x^{(2)}, \ldots, x^{(t+1)})$ independent of $x^{(\tau)}$ where $t \neq \tau$ since there are no hidden recurrences

$$= \log P(y^{(2)} \mid y^{(1)}, x^{(2)}) + \log P(y^{(1)} \mid x^{(1)})$$

Teacher forcing

Teacher forcing can be used in networks with output and hidden layer recurrences, but…

we can no longer parallelize training and must use back-propagation through time

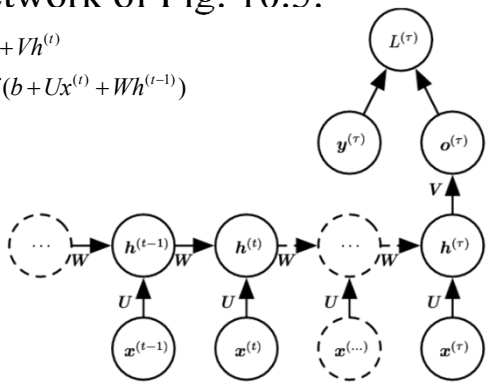


SAN DIEGO STATE UNIVERSITY

RNN gradient computation

- Assume loss at time t: $\partial L / \partial L^{(t)} = 1$
- Consider network of Fig. 10.5:

$$o^{(t)} = c + Vh^{(t)}$$
$$h^{(t)} = f(b + Ux^{(t)} + Wh^{(t-1)})$$



SAN DIEGO STATE UNIVERSITY

20

RNN gradient computation

- Assume o feeds softmax layer $\hat{y}$
- Use negative log likelihood loss function.
  - Our first use of this loss on a softmax layer (we usually use cross entropy)
  - Gradients computed recursively

RNN gradient computation

- In general, the gradient is $\frac{\partial L}{\partial o_i^{(t)}}$
- Consider the last step, time t
  - Define change of overall loss with respect to loss at time t as 1: $\partial L / \partial L^{(t)} \triangleq 1$
  - then

$$\left(\nabla_{o^{(t)}} L\right)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = 1 \frac{\partial L}{o_i^{(t)}} \quad \text{chain rule \&} \frac{\partial L}{\partial L^{(t)}} = 1$$

$$= \frac{\partial L}{o_i^{(t)}} \left(-\log(\hat{y}_i) - -\log(1_{i,y^{(t)}})\right)$$

$$= \frac{\partial L}{o_i^{(t)}} \left(\log\left(\hat{y}_i^{(t)}\right)^{-1} - \log(1_{i,y^{(t)}}^{-1})\right)$$

$$= \hat{y}_i^{(t)} - 1_{i,y^{(t)}}$$

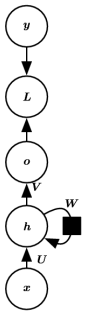
RNN gradient computation

- Given the output gradient at time τ $\nabla_{o^{(\tau)}} L$, compute gradient of the hidden node

$$\nabla_{h^{(\tau)}} L = V \nabla_{o^{(\tau)}} L$$

- Previous time steps $t < \tau$ need to consider history

$$\nabla_{h^{(t)}} L = \left(\frac{\partial h^{(t+1)}}{\partial h^{(t)}} \right)^T \nabla_{h^{(t+1)}} L + \left(\frac{\partial o^{(t+1)}}{\partial h^{(t)}} \right)^T \nabla_{o^{(t)}} L$$



Further simplification depends on activation function derivative

Parameter updates

We need to learn how the loss should be applied to each of the parameters. General idea:

- Create copies of parameters at each time step (avoids introducing dependencies)
- Average over time

Example:

$$\nabla_W L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{W^{(t)}} h_i^{(t)}$$

oh, oh…
rabbit hole

British Postal Service,
Graham Baker-Smith 2015

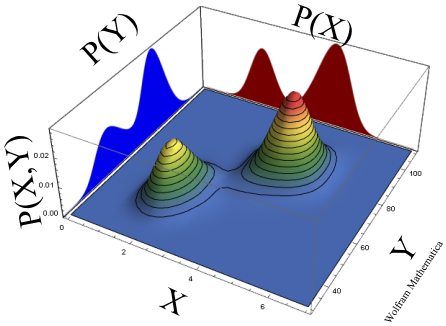
Marginal distributions

Integrate/sums over elements of a joint distribution to find P of desired random variable

$$X, Y \sim P(X, Y)$$

$$P(X = x) = \int_y P(x, y)dy$$

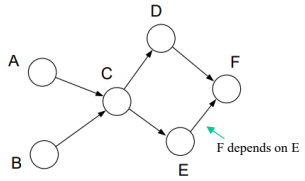
$$P(Y = y) = \int_x P(x, y)dx$$



Graphical models

- Networks of nodes can be used for inference/belief, frequently on latent (unobservable) variables
- Weights inform relationships between nodes
- Allows us to think about probabilities by marginalizing what we don't know

$$P(F \mid A, B) = \frac{\sum_C \sum_D \sum_E P(A, B, C, D, E, F)}{\sum_C \sum_D \sum_E \sum_F P(A, B, C, D, E, F)}$$



Jordan 1997 doi:10.1.1.116.7467

RNNs as directed graphical models

- Simple case: RNN of a sequence where only recursion is previous output and there are *no inputs*

$$P(y^{(1)},\ldots,y^{(\tau)}) = \prod_{t=1}^{\tau} P(y^{(t)} \mid y^{(t)}, y^{(t-1)},\ldots,y^{(2)}, y^{(1)}) \quad \text{by chain rule of P}$$

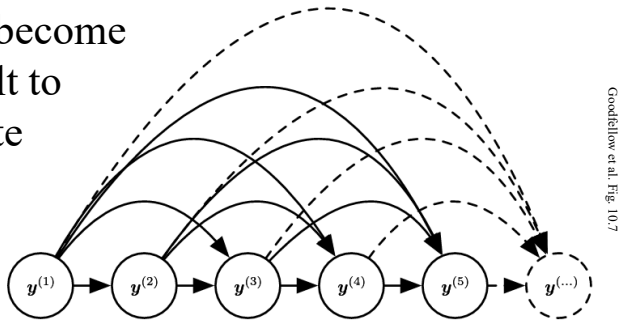
- With a negative log likelihood loss,

$$L = \sum_t L^{(t)}$$

$$L^{(t)} = -logP(\hat{y}^{(t)} = y^{(t)} \mid y^{(t-1)}, y^{(t-2)},\ldots,y^{(1)})$$

RNNs as directed graphical models

- Graphical models with a long dependency require many edges
- These become difficult to estimate



Goodfellow et al. Fig. 10.7

RNNs as directed graphical models

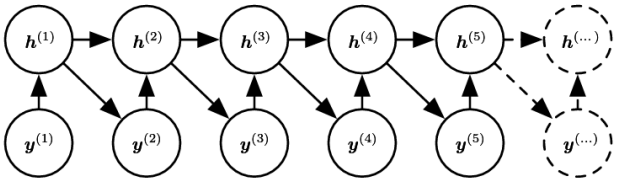
Ways to deal with large # of parameters

- Remove weak dependencies
- Assume a Markov property, values more than τ time steps in the past no longer matter, e.g. τ =2:

$$P(y^{(t)}|y^{(t-1)}, \ldots, y^{(2)}, y^{(1)})$$
$$= P(y^{(t)}|y^{(t-1)}, y^{(t-2)})$$

RNNs as directed graphical models

- Introduce a state variable that captures the dependency



Goodfellow et al. Fig. 10.8

- Assumption: Relationship stationary, does not change over time

RNNs as directed graphical models

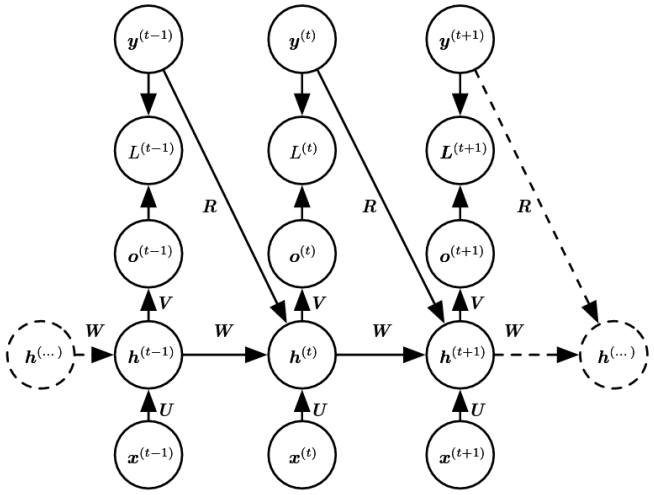
Use as a sequence generator introduces a new problem: How to stop…

- Can introduce a stop symbol, or
- Train a node to predict end of sequence
- Train a node to predict sequence length

Sequences conditioned on context

- RNNs can serve as graphical models that depend on input & output history
- When the input sequence is of fixed size (usually not the case for speech) can provide
  - sequence as input to initial state
  - sequence as input at each time step

Input and output context example
(Fig. 10.10 Goodfellow et al.)



Bidirectional RNNs

- Sometimes, observing features in the future helps us know what occurred in the past…

$$P(x^{(2)} = e \mid x^{(1)} = b)$$

vs.

$$P(x^{(2)} = e \mid x^{(1)} = b, x^{(3)} = d)$$

- Very useful for speech recognition where coarticulation & phonotactics can play a large role

Bidirectional RNNs

- Compute two RNNs, one moving forward, the other backwards
- Avoids need for fixed length window required for feed-forward nets or RNNs with a look-ahead buffer

Bidirectional RNN Example

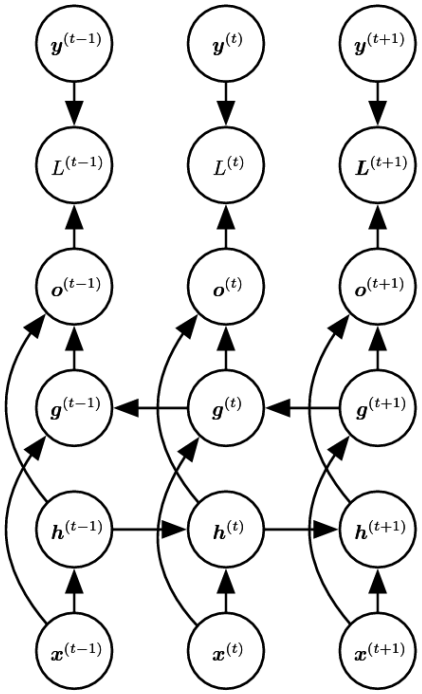
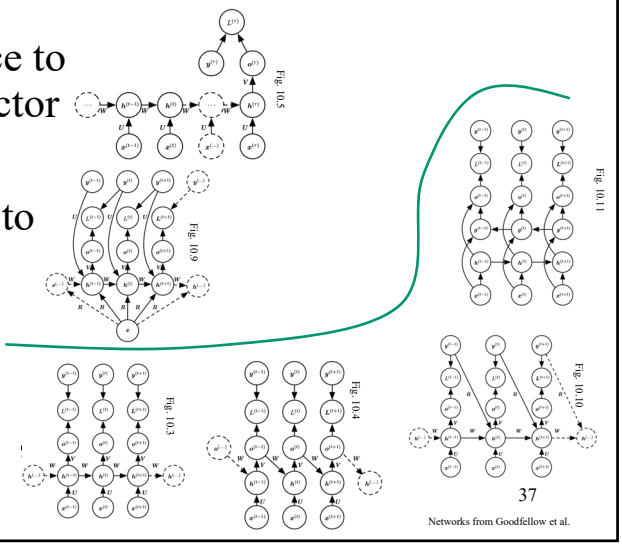


Fig. 10.11 Goodfellow et al.

Recap so far

- Map sequence to fixed size vector
- Fixed vector to sequence
- N inputs to N outputs



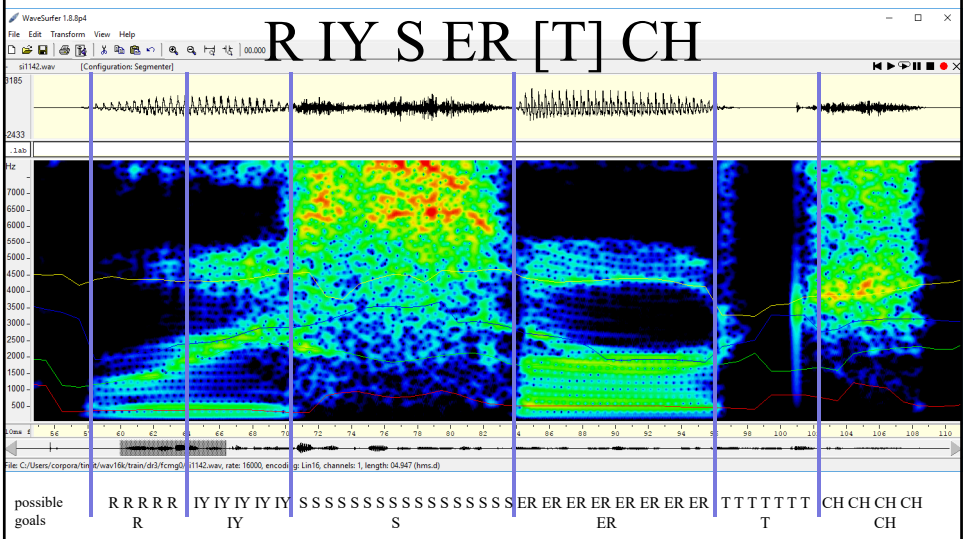
SAN DIEGO STATE UNIVERSITY

37

Networks from Goodfellow et al.

What about this?

R IY S ER [T] CH



possible goals

R R R R R R
IY IY IY IY IY
S S S S S S S S S S S S S S S S S S S S
ER ER ER ER ER ER ER ER
T T T T T T T
CH CH CH CH CH

SAN DIEGO STATE UNIVERSITY

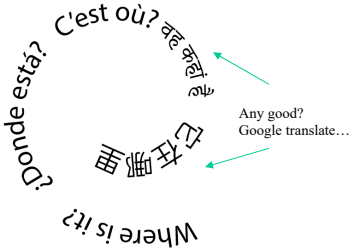
10 ms advance

38

M to N sequences

Many applications require sequences of length M to be mapped to sequences of length N, e.g.

- speech recognition
- language translation



- query handling
  How can I get to Logan Heights?

M to N sequences

- Neural nets that handle this are known as *sequence to sequence*, or *encoder-decoder* architectures

- Basic idea:
  - Encoder RNN outputs a state or sequence representing the *concept*
  - Decoder RNN maps the concept onto another squence

M to N sequences

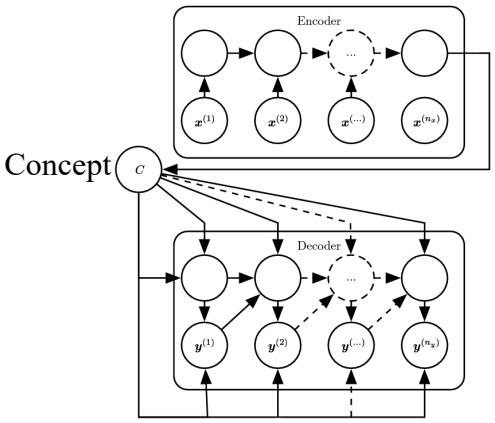


Fig. 10.12 Goodfellow et al.

Concept is typically final state or vector

Can use fixed vector to sequence architecture to generate new sequence

Deep recurrent networks

What happens when the recursions pass through multiple layers?

Layer n (could be multiple layers) serves the role of building a representation to feed into the hidden state layer h

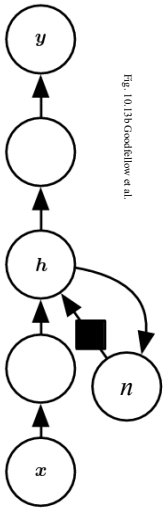


Fig. 10.13b Goodfellow et al.

Deep recurrent networks

- If layer n is deep, this creates a long path between h and the information that loops back through it
- Long paths make learning difficult

  (more on this next)
- Can introduce a delay loop to assist

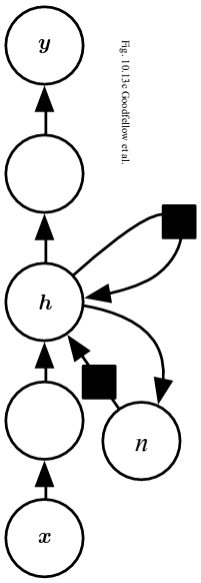


Fig. 10.13c Goodfellow et al.

SAN DIEGO STATE UNIVERSITY

Long-term dependencies

- Recurrence involves computing the same function over and over
- Let us simplify to think about implications
  - Use identity function as activation function
  - Recurrence becomes $h^{(t)} = W^T h^{(t-1)}$
  - Unwinding the recursion $h^{(t)} = (W^t)^T h^{(0)}$

SAN DIEGO STATE UNIVERSITY

44

Long-term dependencies

- Suppose W has an eigenvalue decomposition (2.7)
  $$W = Q\Lambda Q^T$$
  - Q is a matrix of eigenvectors, $QQ^T = I$
  - Λ is a diagonal matrix of eigenvalues

Long-term dependencies

h $= (W^t)^T h^{(0)}$

$= (Q^T \Lambda Q)^t h^{(0)}$ as $W^T = (Q\Lambda Q^T)^T = Q^T \Lambda Q$

$= Q^T \Lambda^t Q h^{(0)}$ as $Q^T \Lambda Q Q^T \Lambda Q = Q^T \Lambda I \Lambda Q = Q^T \Lambda\Lambda Q$

What happens if eigen values are
- small?
- large?

Long-term dependencies

- In a non-recurrent net, weights will vary, but in RNNs the same weights are used over and over again.
- Techniques must be applied to be keep gradients from vanishing or exploding
  (see references in Goodfellow et al 10.7 for further details)

Long-term dependence strategies

Skip connections

- Create networks with longer delays
- Reduces number of times weight is applied
- Delay of d reduces gradient vanishing by factor of $1/d$ (in our simplified example: $Q\Lambda^{t/d}Q^T h^{(0)}$)
- Problems if what was skipped was important

Long-term dependence strategies

- Gated RNNs
  - Long-term dependence problem due to using the same weights at each step
  - What if the weights were dynamic?
  - Dynamic weights allow us to respond to input

Long short-term memory (LSTM)

- Popular gated architecture
- Weights are conditioned on context
- Can be interpreted as an attention mechanism

LSTM

LSTM adds one or more gates to control input and output as well as reset the state

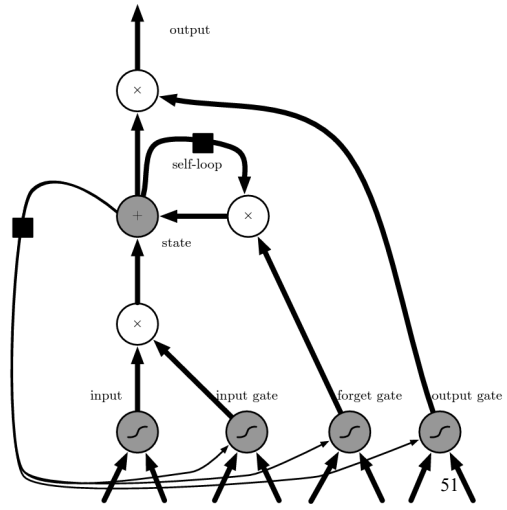


Fig. 10.16 Goodfellow et al.

SAN DIEGO STATE UNIVERSITY

51

LSTM forward propagation

forget gate

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right)$$

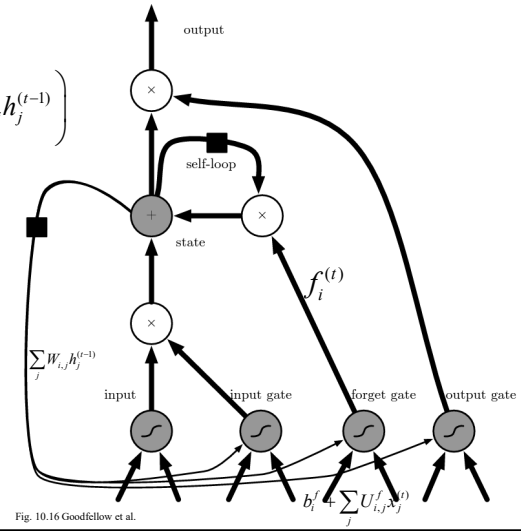


Fig. 10.16 Goodfellow et al.

SAN DIEGO STATE UNIVERSITY

LSTM forward propagation

external input gate

$$g_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^{(o)} h_j^{(t-1)}\right)$$

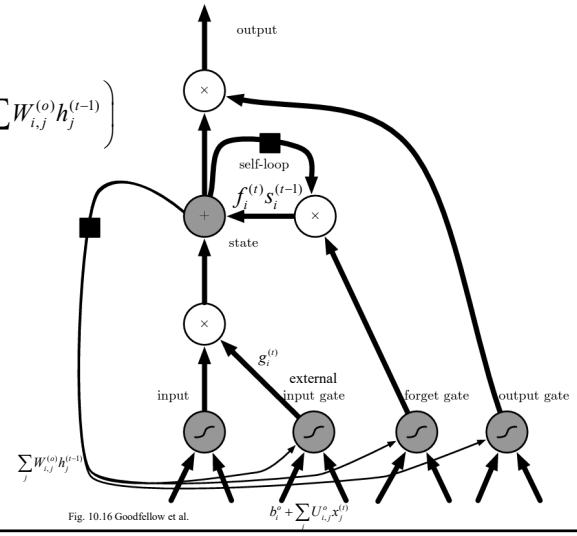


Fig. 10.16 Goodfellow et al.

SAN DIEGO STATE UNIVERSITY

LSTM forward propagation

state update

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right)$$

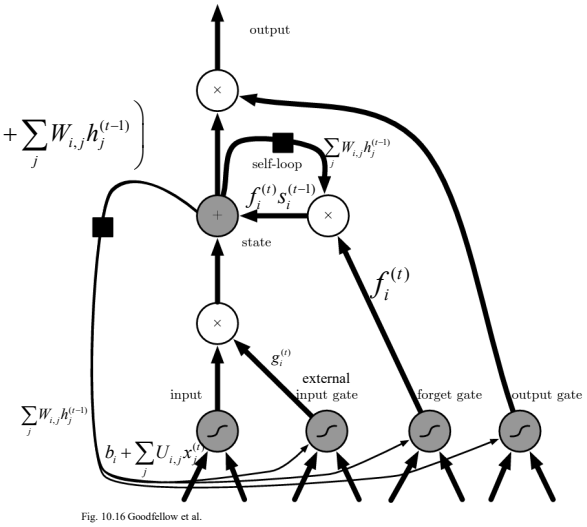


Fig. 10.16 Goodfellow et al.

SAN DIEGO STATE UNIVERSITY

LSTM forward propagation

output

$$h_i^{(t)} = \tanh(s_i^{(t)}) q_i^{(t)}$$

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)} \right)$$

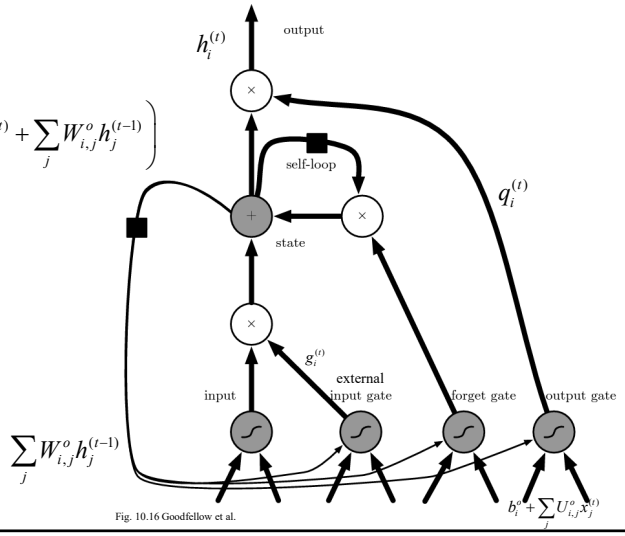


Fig. 10.16 Goodfellow et al.

SAN DIEGO STATE UNIVERSITY