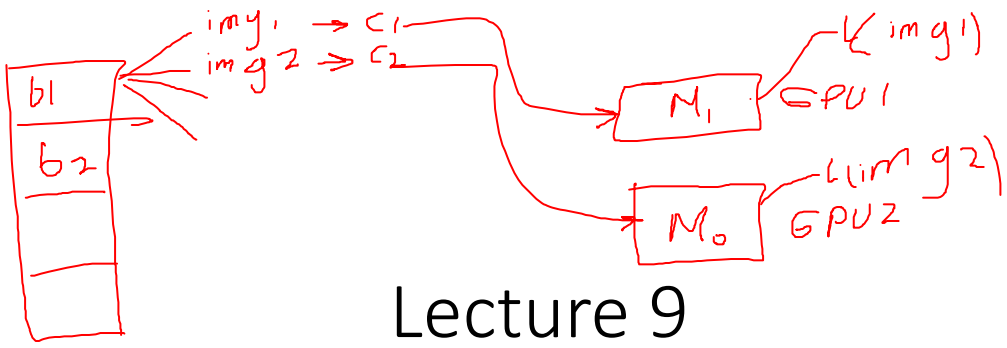


Lecture 9

Deep learning

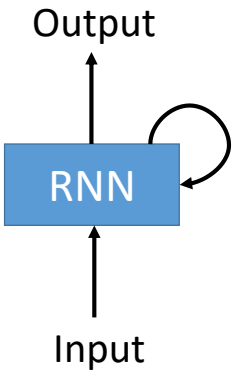
1

Recurrent Neural Network

- **The recurrence formula**

$$h^{(t)} = f\left(h^{(t-1)}, x^{(t)}; \theta\right)$$

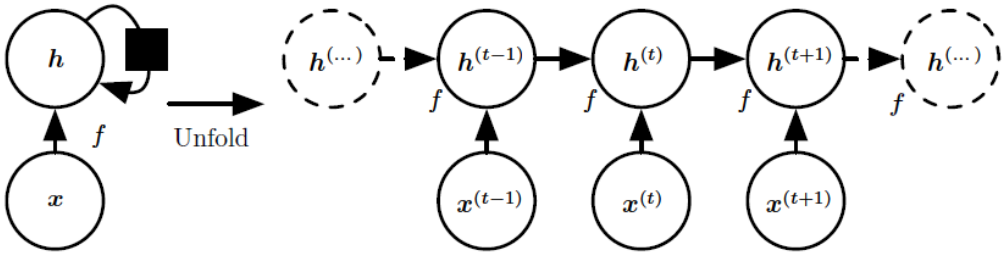
- x(t) an input sequence at time t
- Same function and same set of parameters are used at each time step



Goodfellow, Bengio, Courville 2016

2

Unfolding

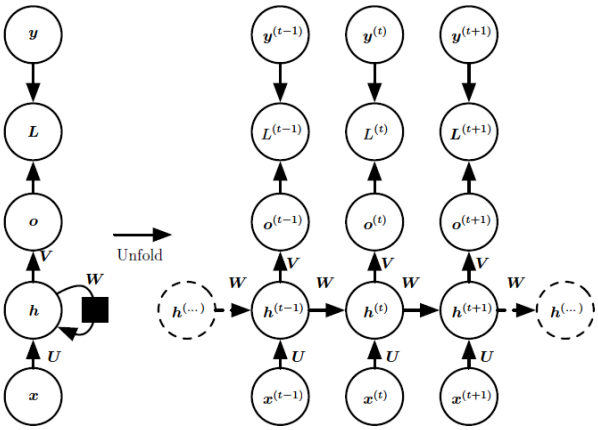


Goodfellow, Bengio, Courville 2016

3

Example Design patterns: Recurrent Hidden Units

- Recurrent networks that produce an output at each time step and have recurrent connections between hidden units (Plain Vanilla RNN)
- *L:* loss, *U:* input to hidden connections parameters, *W:* hidden-to-hidden recurrent connections parameters, and *V:* hidden-to-output connections parameters
- maps an input sequence to an output sequence of the same length.
- Powerful networks: can choose to put any information it wants about the past into its hidden representation *h* and transmit *h* to the future.



Goodfellow, Bengio, Courville 2016

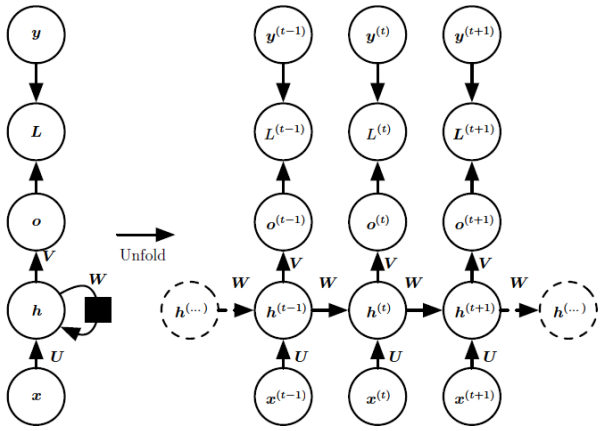
4

Feedforward Propagation: Plain Vanilla RNN

- Assume discrete output (e.g. words, characters) and hyperbolic tangent activation function

$$\begin{aligned} \boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\ \boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\ \boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\ \hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}) \end{aligned}$$

- *o* as giving the unnormalized log probabilities of each possible value of the discrete variable.
- We apply the softmax operation to obtain a vector $\hat{y}$ of normalized probabilities over the output.



Goodfellow, Bengio, Courville 2016

5

Feedforward Propagation

- The total loss for a given sequence of *x* values paired with a sequence of *y* values would then be just the sum of the losses over all the time steps.

- The gradient computation involves performing a forward propagation pass moving left to right, followed by a backward propagation pass moving right to left through the graph.
- The runtime is O(τ) and cannot be reduced by parallelization because the forward propagation graph is inherently sequential

$$\begin{aligned} & L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right) \\ &= \sum_t L^{(t)} \\ &= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right), \end{aligned}$$

Goodfellow, Bengio, Courville 2016

6

Backward Propagation

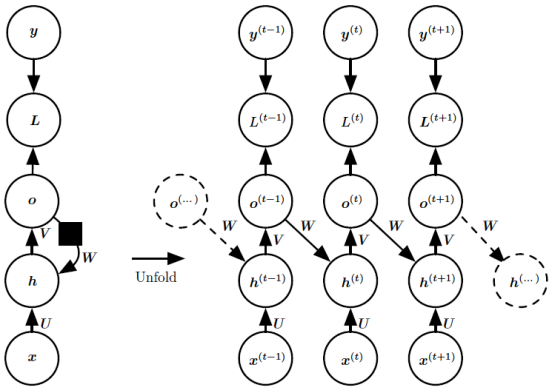
- States computed in the forward pass must be stored until they are reused during the backward pass, so the memory cost is also $O(\tau)$.
- The back-propagation algorithm applied to the unrolled graph with $O(\tau)$ cost is called **back-propagation through time** or **BPTT**
- The network with recurrence between hidden units is thus very powerful but also expensive to train.
- We need to compute *U , V , W (in addition to biases b,c) (*we use gradient-based techniques)
- We need to compute the gradient $\nabla_U L, \nabla_V L, \nabla_W L$
- We move from right to left (i.e. start from the O(t)), backpropagation through time.

Goodfellow, Bengio, Courville 2016

7

Example Design patterns: Recurrence through only the output

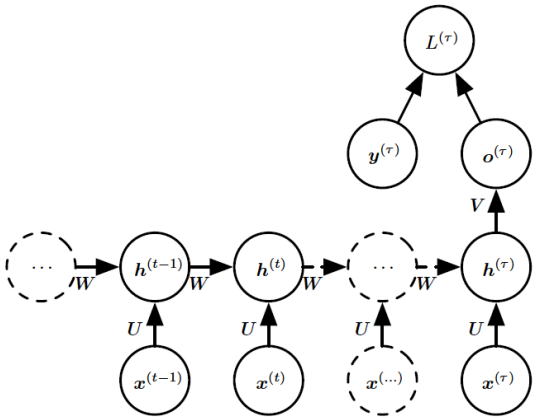
- Recurrent networks that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step.
- Less powerful (can express a smaller set o functions) than those in the family represented by the recurrent hidden units.
- Easier to train
- Trained to put a specific output value into *o* , and *o* is the only information it is allowed to send to the future.
- Unless *o* is very high-dimensional and rich it will usually lack important information from the past.



Goodfellow, Bengio, Courville 2016

8

Example Design patterns: Sequence Input, Single Output



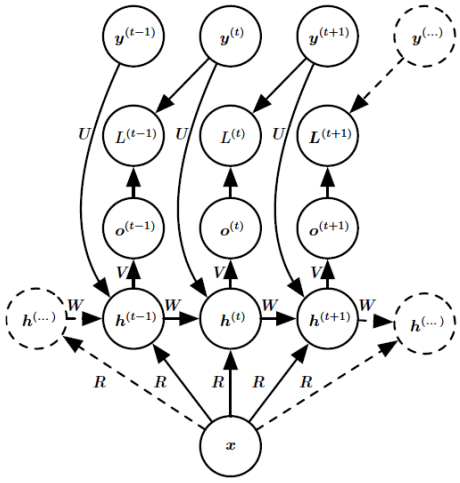
- Can be used to summarize a sequence and produce a fixed-size representation.

Goodfellow, Bengio, Courville 2016

9

Example Design patterns: Vector to Sequence

- RNNs can take only a single vector *x* as input.
- When *x* is a fixed-size vector, we can simply make it an extra input of the RNN that generates the y sequence. Some common ways of providing an extra input to an RNN are:
  1. as an extra input at each time step (most common), or
  2. as the initial state *h*(0), or
  3. both.
- The interaction between the input *x* and each hidden unit vector $h^{(t)}$ is parametrized by a newly introduced weight matrix *R* that was absent from the model of only the sequence of *y* values. The same product $x^T R$ is added as additional input to the hidden units at every time step.
- Application: Image Captioning



Goodfellow, Bengio, Courville 2016

10

Knowing When to Look: Adaptive Attention via A Visual Sentinel for Image Captioning. Jiasen Lu, Caiming Xiong, Devi Parikh, Richard Socher

11

Example Design patterns: Bidirectional RNNs

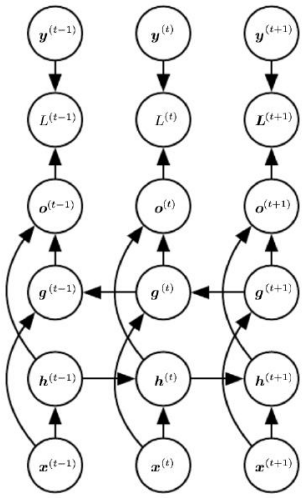
- All of the recurrent networks we have considered up to now have a "causal" structure, meaning that the state at time t only captures information from the past, $x^{(1)}, \ldots, x^{(t-1)}$, and the present input $x^{(t)}$.
- In many applications we want to output a prediction of $y^{(t)}$ which may depend on *the whole input sequence (including future)*.
- For example, in speech recognition: if there are two interpretations of the current word that are both acoustically plausible, we may have to look far into the future (and the past) to disambiguate them.
- This is also true of handwriting recognition and many other sequence-to-sequence learning tasks.

Goodfellow, Bengio, Courville 2016

12

Bidirectional RNNs

- As the name suggests, bidirectional RNNs combine an RNN that moves forward through time beginning from the start of the sequence with another RNN that moves backward through time beginning from the end of the sequence.
- $h^{(t)}$standing for the state of the sub-RNN that moves forward through time and $g^{(t)}$ standing for the state of the sub-RNN that moves backward through time.
- This allows the output units *o*(*t*) to compute a representation that depends on *both the past and the future.*
- Applications: speech recognition, bioinformatics, handwriting recognition

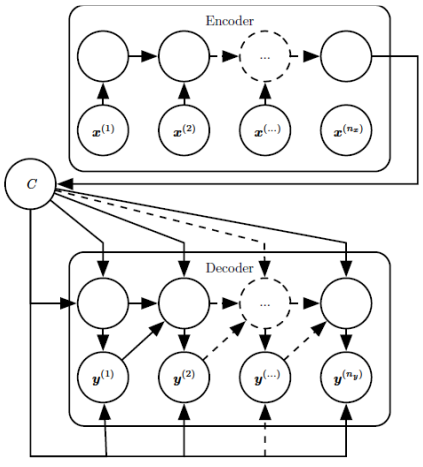


Goodfellow, Bengio, Courville 2016

13

Encoder-Decoder Sequence-to-Sequence Architectures

- Used to map an input sequence to an output sequence which is not necessarily of the same length (e.g. machine translation, question answering,).
- *C:* A representation of the input (Context)
- Has two components:
  - an **encoder** or **reader** or **input** RNN: processes the input sequence and produces the context *C* , usually as a simple function of its final hidden state.
  - a **decoder** or **writer** or **output** RNN is conditioned on that fixed-length vector to generate the output sequence



Goodfellow, Bengio, Courville 2016

14

Deep Recurrent Networks

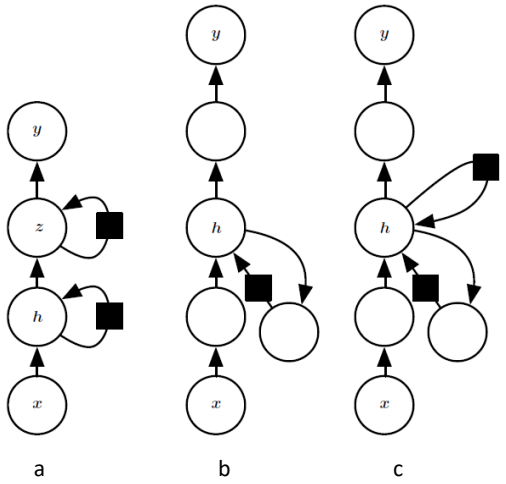
- The computation in most RNNs can be decomposed into three blocks of parameters and associated transformations:
  1. from the input to the hidden state,
  2. from the previous hidden state to the next hidden state, and
  3. from the hidden state to the output.
- When the network is unfolded, each of these corresponds to a shallow transformation (a transformation that would be represented by a single layer within a deep MLP).
- Experimental evidence by(Graves *et al.*, 2013; Pascanu *et al.*, 2014a) strongly suggests that introducing depth in these operations would be advantageous.
- Adding depth, however, will make the optimization much more difficult

Goodfellow, Bengio, Courville 2016

15

Deep Recurrent Networks

- *"How to Construct Deep Recurrent Neural Networks"* (Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio)
- A recurrent neural network can be made deep in many ways (Pascanu *et al.*,). The hidden recurrent state
  - can be broken down into groups organized hierarchically. *(a)*
  - Deeper computation (e.g., an MLP) can be introduced in the input-to-hidden, hidden-to-hidden and hidden-to-output parts. This will lengthen the shortest path linking different time steps. (b)
  - The path-lengthening effect can be mitigated by introducing skip connections.(c)

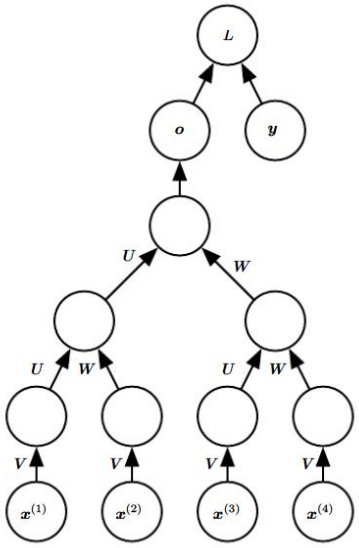


Goodfellow, Bengio, Courville 2016

16

Recursive Neural Networks

- The computational graph is structured as a deep tree, rather than the chain-like structure.
- Recursive networks have been successfully applied in natural language processing (Socher *et al.*, 2011a,c, 2013a) as well as in computer vision (Socher *et al.*, 2011b).
- One advantage of recursive nets over recurrent nets is that for a sequence of the same length τ, the depth (measured as the number of compositions of nonlinear operations) can be drastically reduced from τ to $O(\log \tau)$, which might help deal with long-term dependencies.
- An open question is how to best structure the tree.



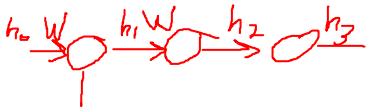
Goodfellow, Bengio, Courville 2016

17

The Challenge of Long-Term Dependencies

- Gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely, but with much damage to the optimization).
- Even if we assume that the parameters are such that the recurrent network is stable (can store memories, with gradients not exploding), the difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions (involving the multiplication of many Jacobians) compared to short-term ones.
- The problem was rst analyzed by Hochreiter and Schmidhuber 1991 and Bengio et al 1993

$h_1 = W h_0 \qquad h_2 = W h_1 = W^2 h_0 \qquad h_3 = W^3 h_0$



Goodfellow, Bengio, Courville 2016

18

The Challenge of Long-Term Dependencies

- the function composition employed by recurrent neural networks somewhat resembles matrix multiplication. We can think of the recurrence relation

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^{\top}\boldsymbol{h}^{(t-1)}$$

- This recurrence relation essentially describes the power method. It may be simplified to

$$\boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^{t}\right)^{\top}\boldsymbol{h}^{(0)}$$

- if *W* admits an eigen decomposition of the form:

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top} \qquad \boldsymbol{h}^{(t)} = \boldsymbol{Q}^{\top}\boldsymbol{\Lambda}^{t}\boldsymbol{Q}\boldsymbol{h}^{(0)}$$

- The eigenvalues are raised to the power of *t* causing eigenvalues with magnitude less than one to decay to zero and eigenvalues with magnitude greater than one to explode.
- Solution: LSTM, GRUs

Goodfellow, Bengio, Courville 2016

19