UNIVERSITÄT BERN

Deep Recurrent Neural Networks

Paolo Favaro

Contents

- Deep Recurrent Neural Networks
  - Modeling sequences, teacher forcing, backpropagation, bidirectional RNNs, deep RNNs, long-term dependencies, echo state nets, LSTMs/ GRUs
- Based on **Chapter 10** of Deep Learning by Goodfellow, Bengio, Courville
- Credits also to the Stanford Course CS231n by Fei-Fei Li, Justin Johnson, Serena Yeung

Note

Sequence Modeling

- While convolutional neural networks are used for data on a grid, **recurrent neural networks (RNN)** are used for **sequential** data

- Based on parameter sharing (across time)

RNNs can scale well with sequential data (sequences with long lengths will not require a large network).
Analogous to CNNs that may process images of different sizes.

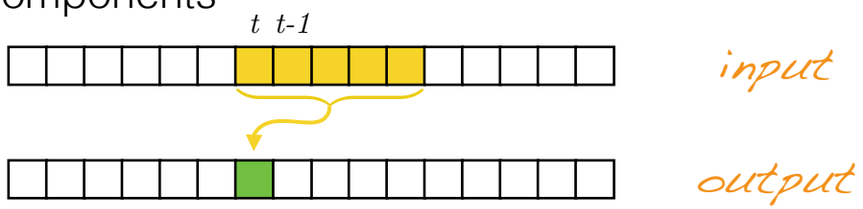
Sequence Modeling

- Example: What year does it refer to?
  - Input #1: “I went to Nepal in 2009.”
  - Input #2: “In 2009, I went to Nepal.”
- A feedforward network with fixed length inputs may need all possible orderings to learn the task; it would use different weights for words at different positions
- Translation invariance is a desirable property

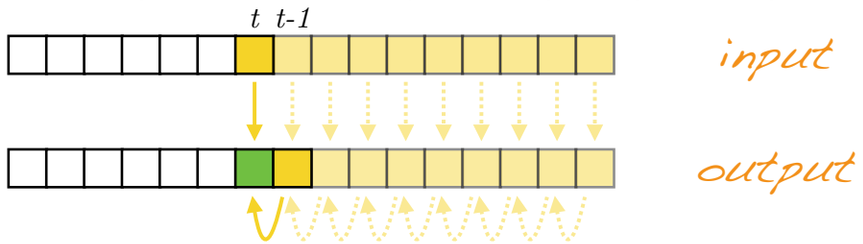
Notes

Sequence Modeling

- 1D convolution: output is a fixed function of a few input components



- RNNs: output is a fixed function of the previous output and current input (and thus possibly all prior input)



A related idea is a 1D convolution. Sharing of weights. At each time instant the calculation involves only a few local input elements (from the sequence). The dependence is limited to a fixed scope.
Instead in the case of the RNNs the scope can potentially be all the past history.

Sequence Modeling

- RNNs define **dynamical systems** described by

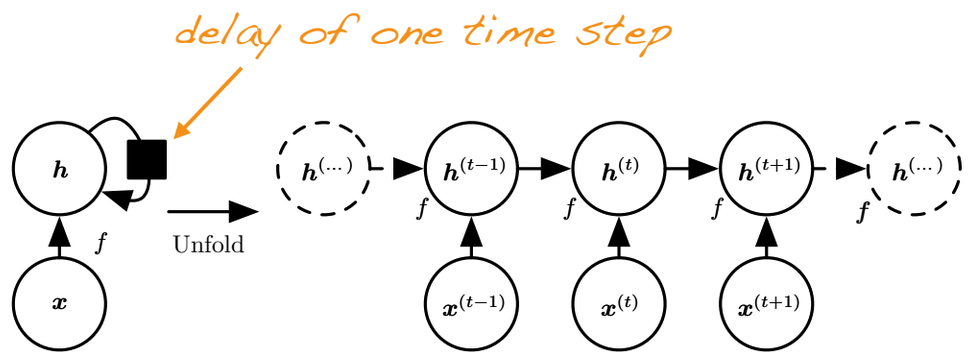
$$h^{(t)} = f\left(h^{(t-1)}, x^{(t)}; \theta\right)$$

where h is the state of the system, x is the input, and θ are the parameters of the network

Notes

Unfolding Computational Graphs

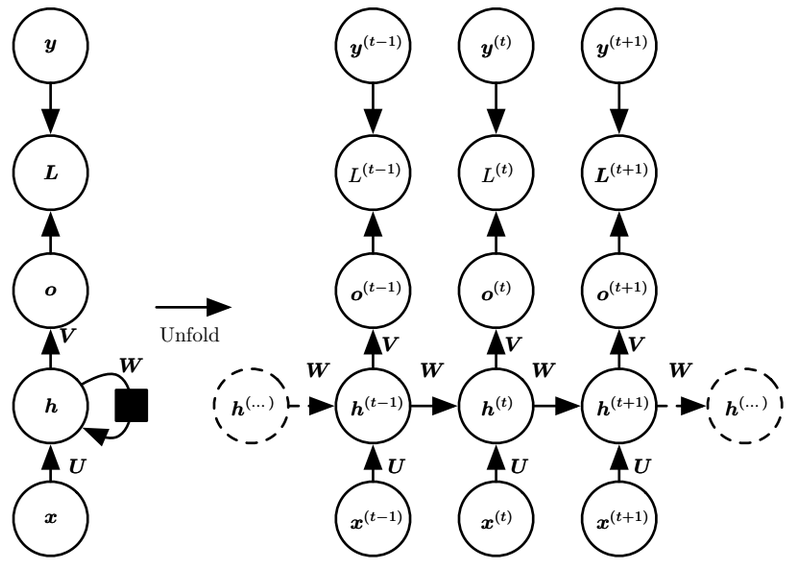
- We can describe RNNs with a graph containing cycles



This is an RNN without outputs. The black box denotes a delay of 1 time step.
Notice that the state h is a function of all past inputs. So one should redefine the function any time a new input is introduced. By using the recurrent form one gets two advantages: 1) the same function can be reused for all inputs (all lengths and instants); 2) the calculations can be done more efficiently (no need to explicitly process all the history all the time).

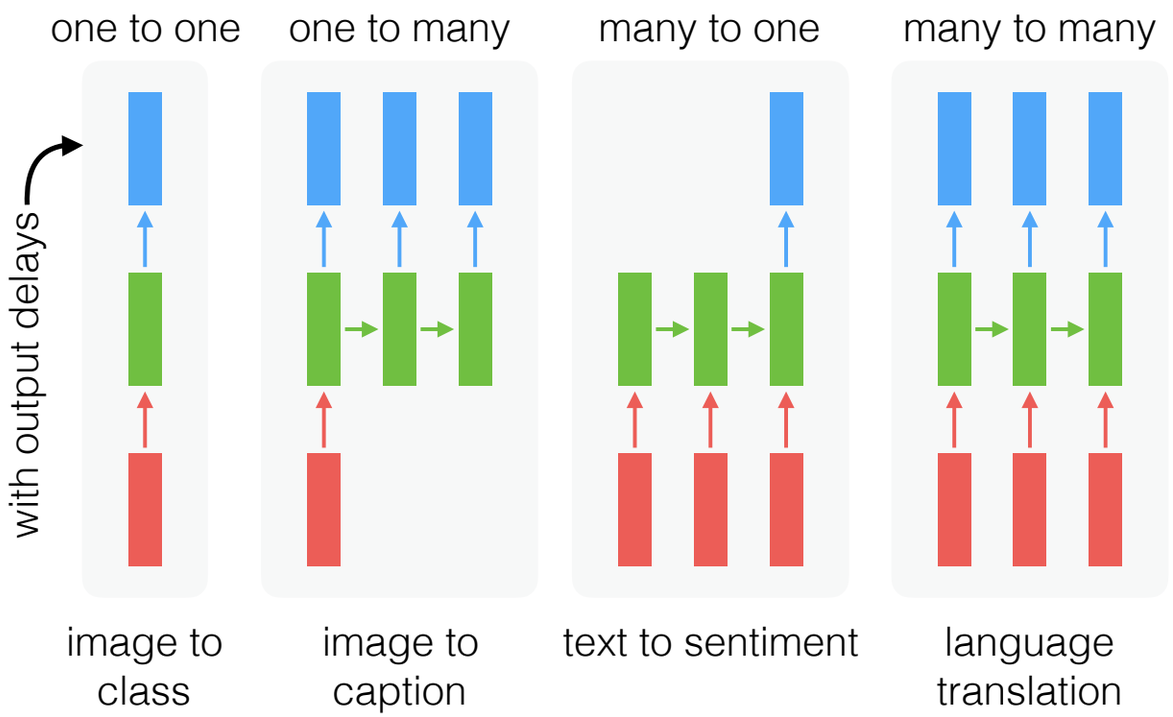
Recurrent Neural Networks

- RNNs also produce an output

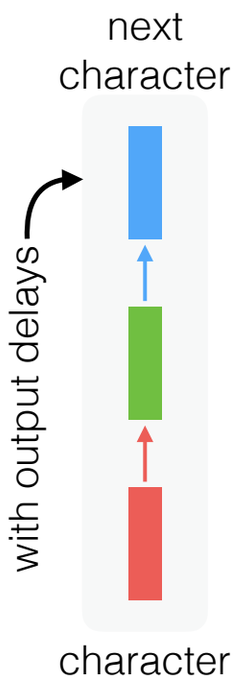


In this implementation the network produces an output o which is then compared against a loss L with the label y (for each time instant). Can simulate a Turing machine.

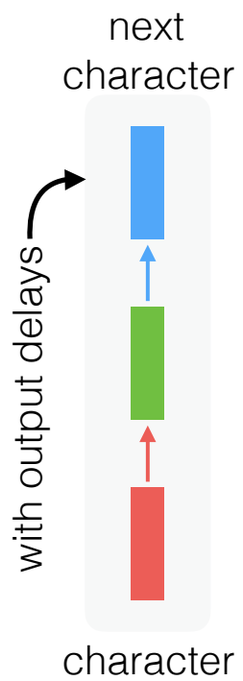
RNN Sequence Mappings



RNN Example I

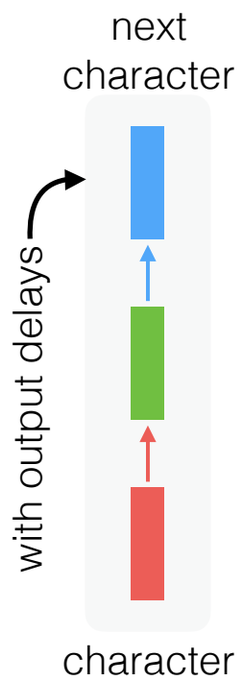


RNN Example I

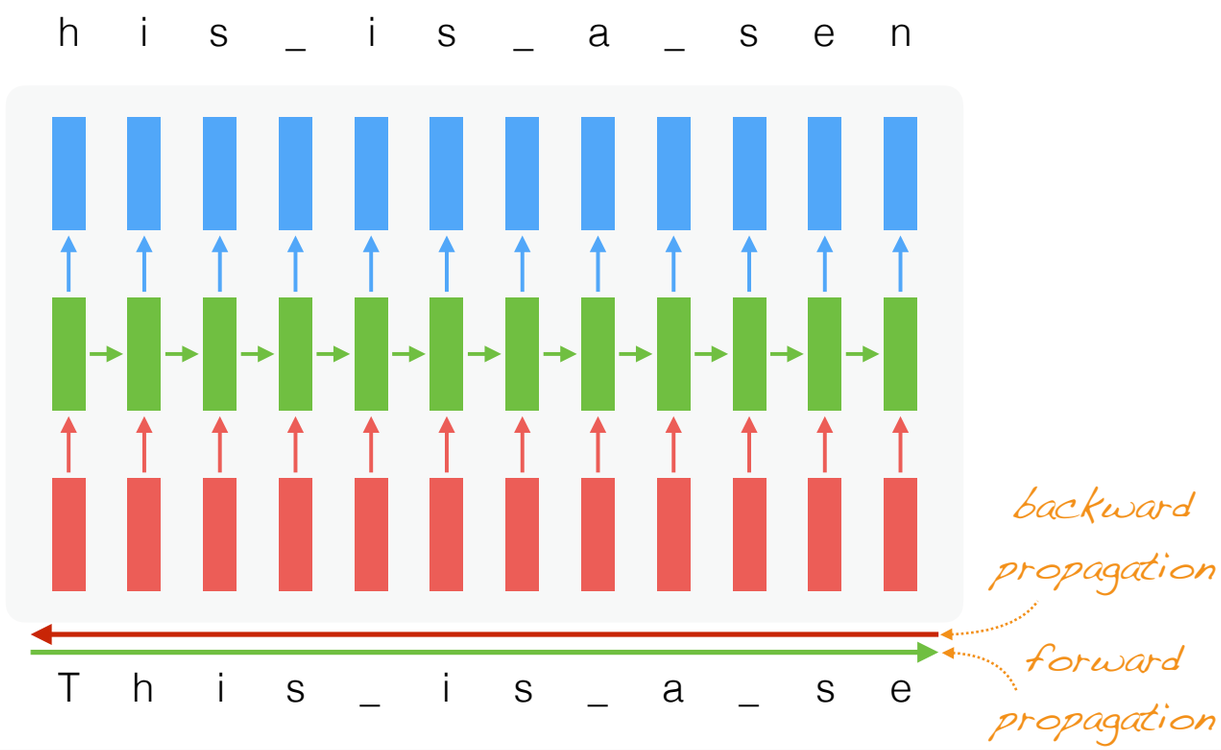


RNN Example I

- **Example**: Character-level language model
- Feed one character at a time and predict the following one (output the probability of each possible character)
- During training use the ground truth as input and in the loss function
- At test time feed output (most likely character) back to the input

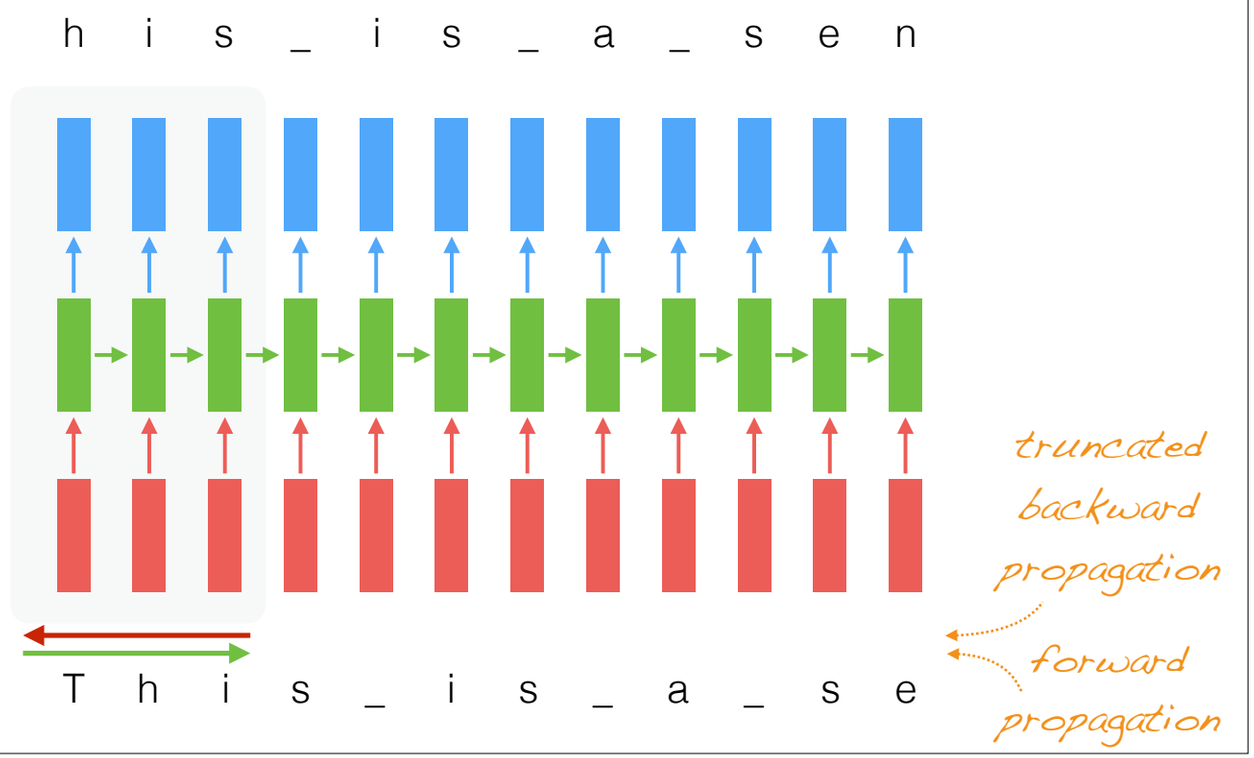


RNN Example I



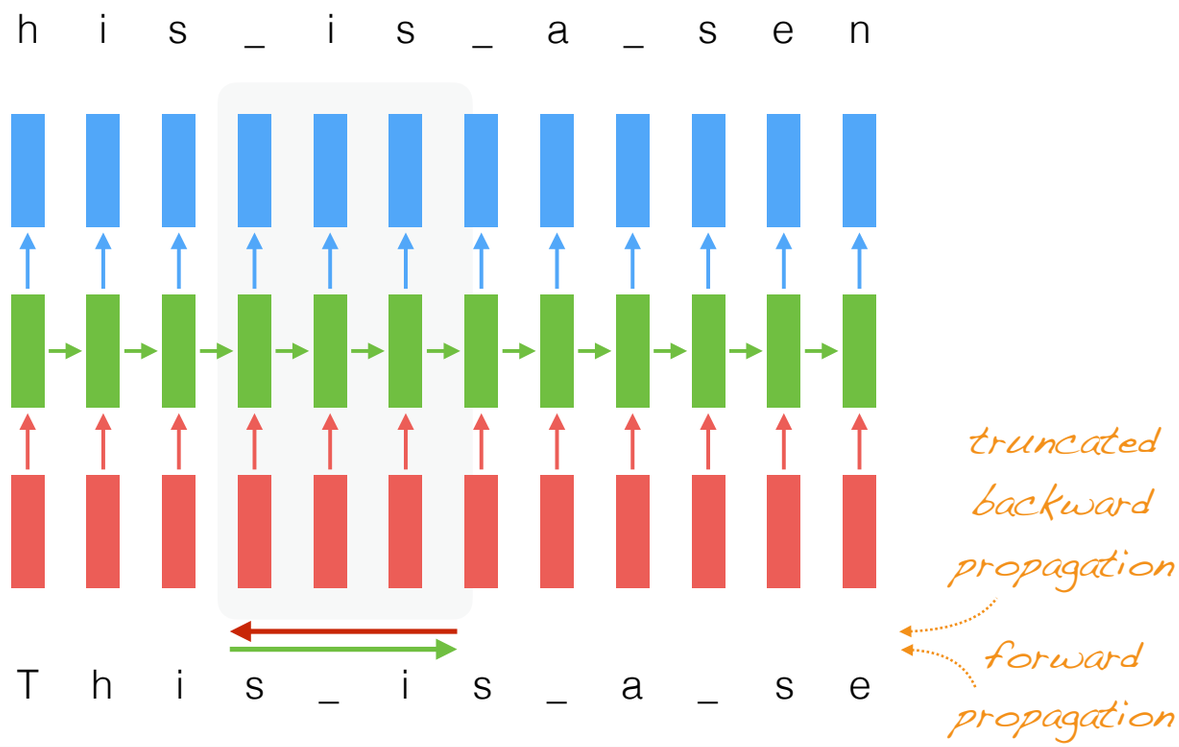
The training of this unrolled network requires feeding the entire sequence up to the period (end of sentence) and then backpropagating. This becomes quickly unfeasible because of memory requirements.

RNN Example I



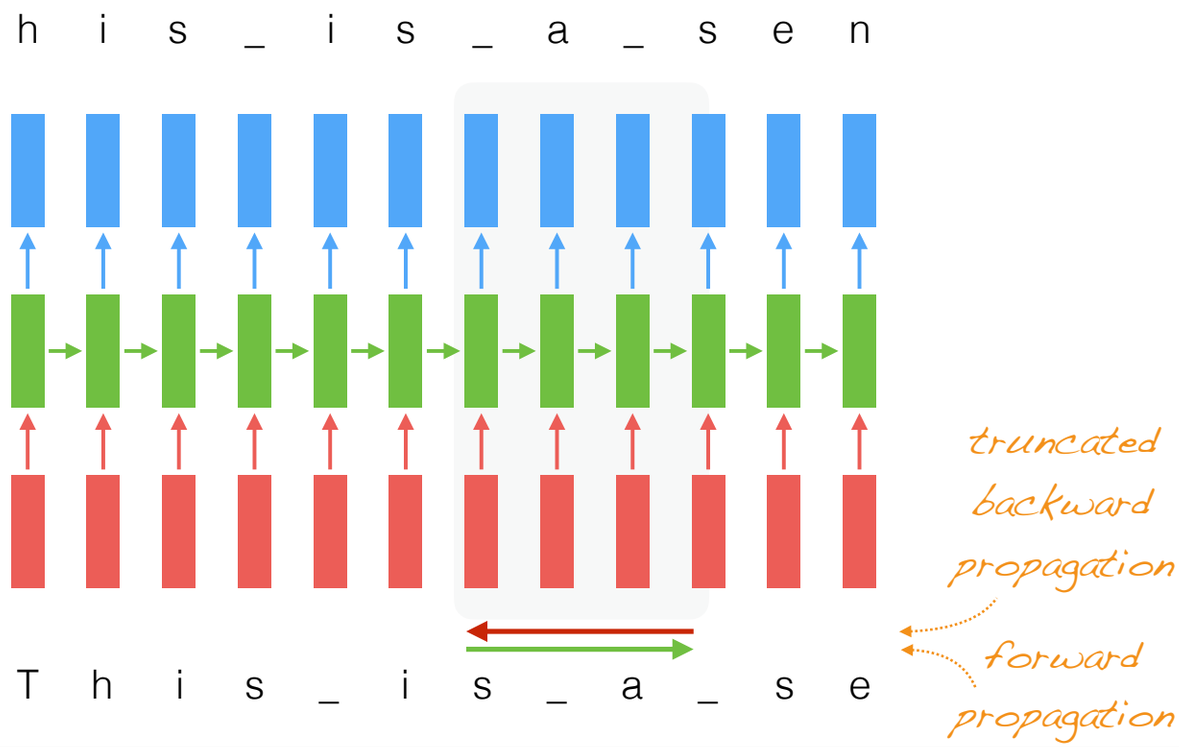
One approach to handle this complexity is to propagate information only block-wise. The forward pass is achieved by carrying forward the hidden state. The backward pass is approximate by being limited to the block size.

RNN Example I



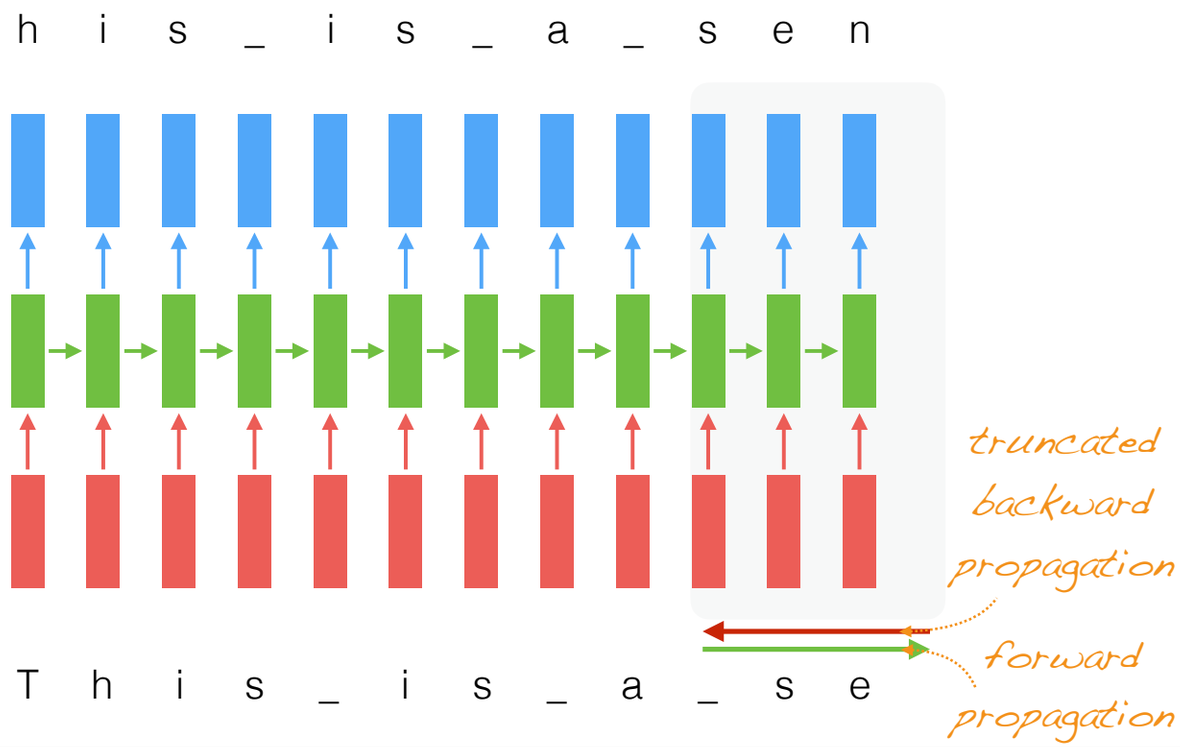
One approach to handle this complexity is to propagate information only block-wise. The forward pass is achieved by carrying forward the hidden state. The backward pass is approximate by being limited to the block size.

RNN Example I



One approach to handle this complexity is to propagate information only block-wise. The forward pass is achieved by carrying forward the hidden state. The backward pass is approximate by being limited to the block size.

RNN Example I



One approach to handle this complexity is to propagate information only block-wise. The forward pass is achieved by carrying forward the hidden state. The backward pass is approximate by being limited to the block size.

RNN Example I

Sample code: min-char-rnn.py

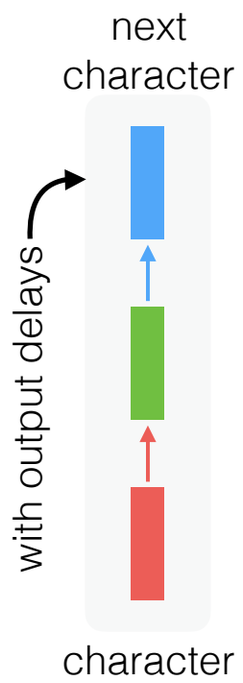
https://gist.github.com/karpathy/d4dee566867f8291f086

RNN Example I

- Train with:

The Sonnets by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.



RNN Example I

```
tyntd-iathatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

RNN Example I

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain d into beirg never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begur out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the war
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaves of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds.
So drop upon your lordship's head, and your opinion
Shall be against your honour.

RNN Example II

- Image captioning
- Task: Given an image generate the corresponding caption
- Use (lots of) examples of images with captions

A baseball player with a glove is throwing a ball.

Image Captioning

- Some literature
  - Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
  - Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
  - Show and Tell: A Neural Image Caption Generator, Vinyals et al.
  - Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
  - Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Image Captioning

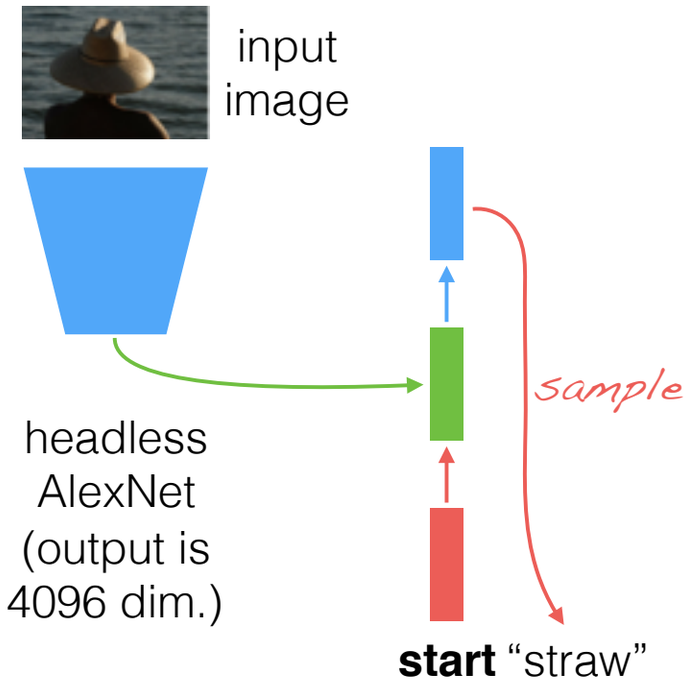


Image Captioning

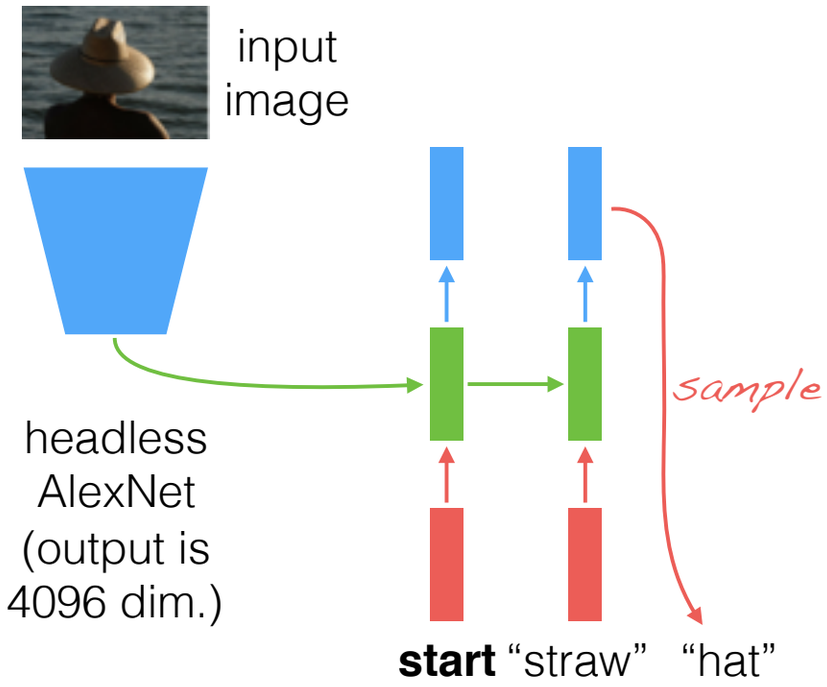


Image Captioning

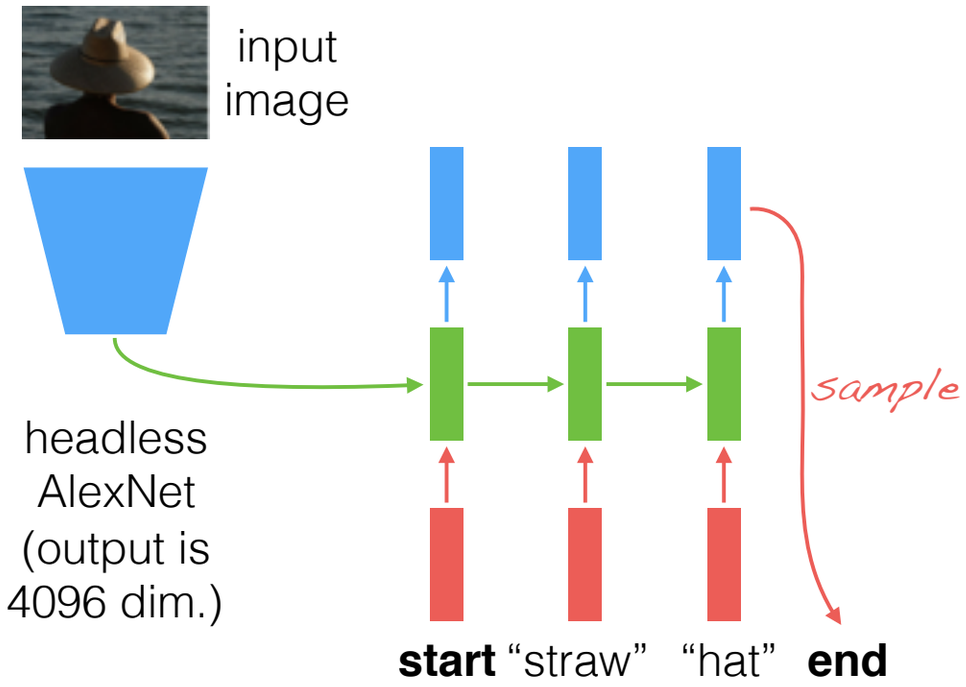


Image Captioning Results

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

Recurrent Neural Networks

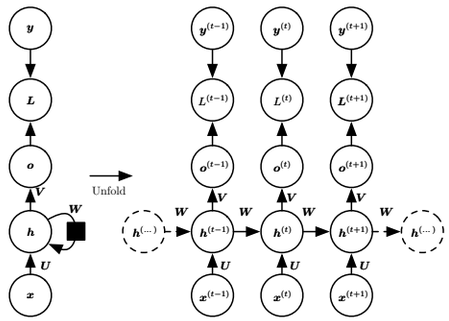
- Discrete output RNN

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

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

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

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



probability vector

with loss $L(\{x^{(1)}, \ldots, x^{(\tau)}\}, \{y^{(1)}, \ldots, y^{(\tau)}\})$

$$= -\sum_t \log p_{\text{model}} \left(y^{(t)} | \{x^{(1)}, \ldots, x^{(\tau)}\} \right)$$

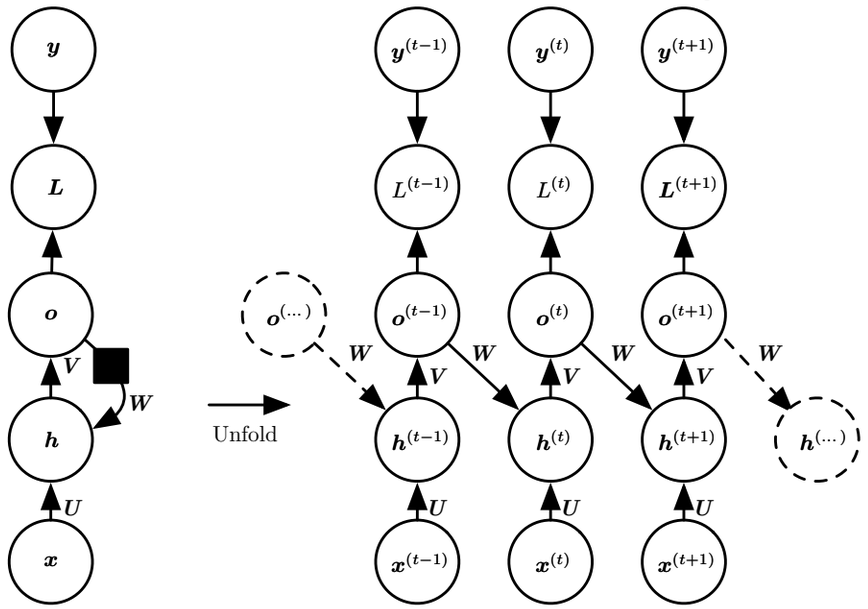
$$= -\sum_t \log \hat{y}_{y^{(t)}} \left(y^{(t)} | \{x^{(1)}, \ldots, x^{(\tau)}\} \right)$$

selected entry

Training is very computationally expensive. Backpropagation through time requires the sequential execution of each step (no parallelization is possible).

Recurrent Neural Networks

- In this case the recurrence can come only from the output



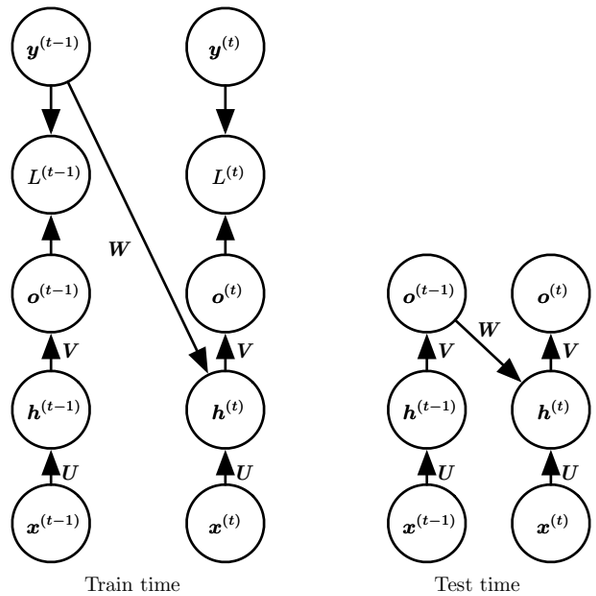
In this case the network is less powerful than before.
It is unlikely that the user knows a representation of the outputs that will be useful also as a hidden state (so that it stores all the useful information).

Teacher Forcing

- **Teacher forcing** consists in feeding the ideal output directly to the hidden state instead of the computed output at the previous iteration
- This scheme allows parallel training (no need to wait for the output and the output is also the state)
- At test time we feed instead the computed output

Note

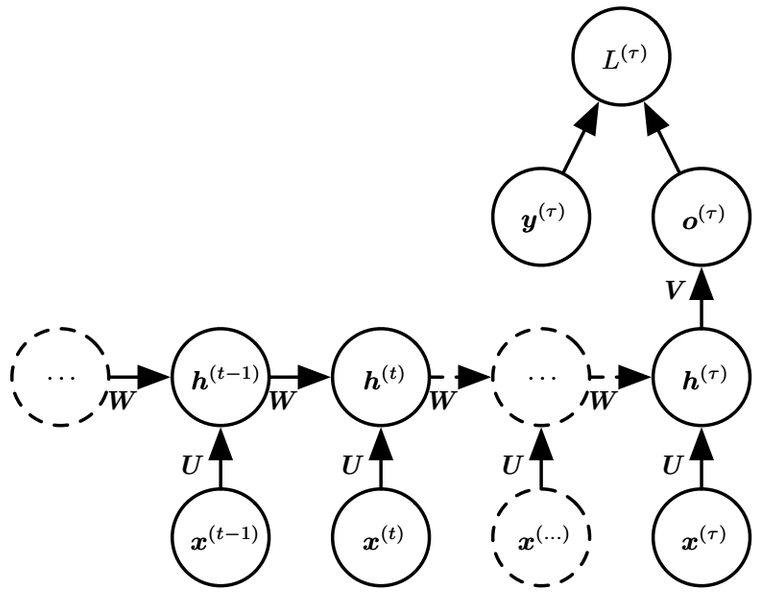
Teacher Forcing



To minimize the difference between what the networks experiences during training with during testing, we can mix the methods. We can run the network for some time with its own output and then use the target output.

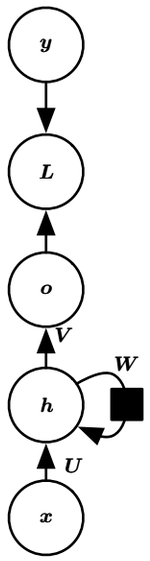
Recurrent Neural Networks

- RNNs with a single final output



Some networks can have a single output at the end of the sequence (and that is the only one checked against the target y).

Computing the Gradient in an RNN



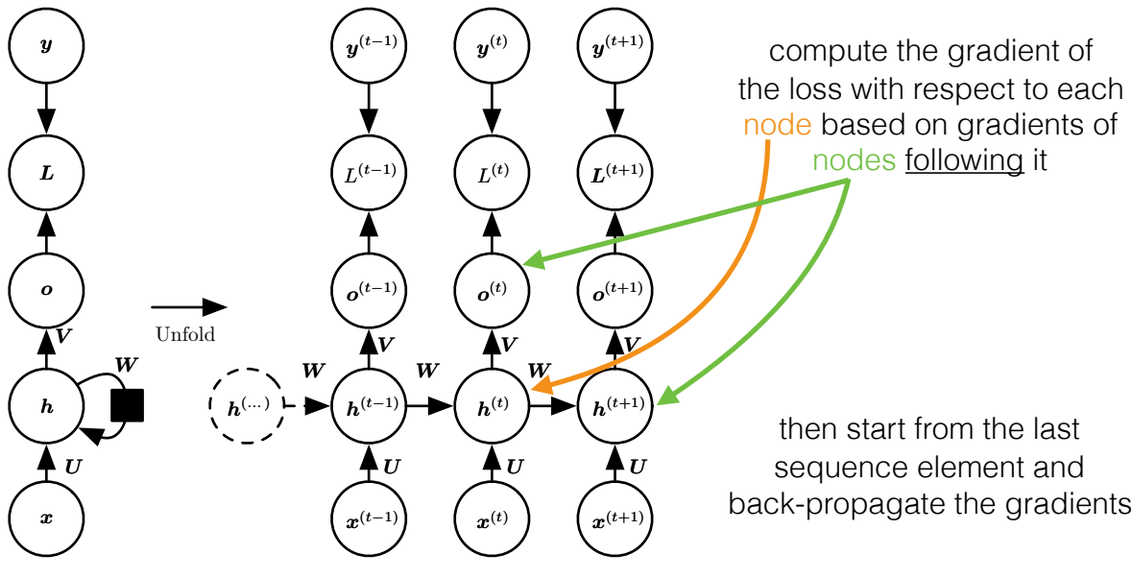
recursion

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

$$\hat{y}^{(t)} = g\left(h^{(t)}; \theta\right)$$

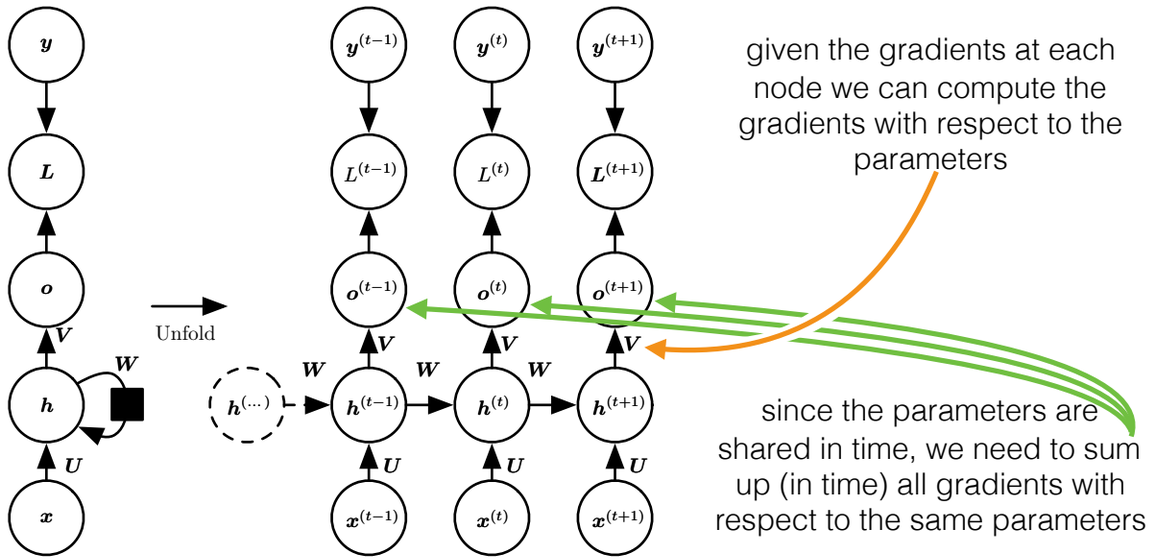
It seems that running SGD on an RNN is hopeless, or at least quite complicated, because it involves a recursive equation.

Computing the Gradient in an RNN



However, it can be computed simply by unrolling the computational graph and by applying the usual back-propagation to it. Say that t+1 is the last element in the sequence. Then h^(t+1) is connected only to the output o^(t+1) (and not another future hidden state). Thus, we can compute the derivatives of the output o^(t+1) with respect to h^(t+1) and then work backwards in time.
Keep in mind that L = \sum_t L^t so dL/dL^t = 1.

Computing the Gradient in an RNN



Note

RNNs as Generative Models

- If we sample an input and feed it to an RNN, the RNN can give us a sequence of output samples
- How to define the length of the output sequence?
  - Add a symbol to define the End Of Sequence (added in training at the end of a sequence)
  - Introduce a Bernoulli output to model the decision to stop (more general)
  - Introduce output to predict the sequence length

The Bernoulli output needs to predict whether the sequence stops or not. If predicting directly the length of the sequence it is better to add as input the predicted length \tau - t so that the RNN knows when the sequence is about to end.

Conditioning on Context

- Consider the **captioning problem**: Given an image we would like to generate a sequence of words.

"baseball player is throwing ball in game."

"woman is holding bunch of bananas."

"black cat is sitting on top of suitcase."

- Thus, we would like to learn a distribution $P(y|x)$ where y is a text sequence and x an image

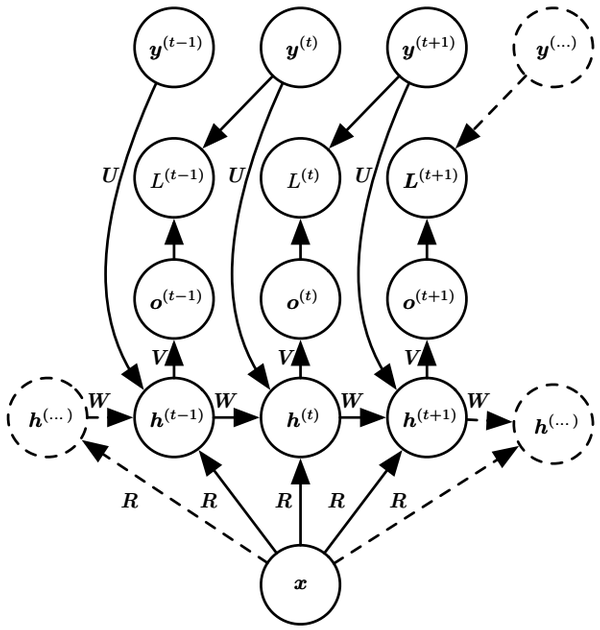
Note

Conditioning on Context

- The image x is the **context** to the text sequence y
- An RNN with such context could be defined as
  1. taking x as the same input at each time step
  2. taking x as the initial hidden state
  3. both of the above

Note

Conditioning on Context



This is an example of the first approach (feeding the same input x at every time instant).

Bidirectional RNNs

- So far we have seen only RNNs with a specific temporal direction (current state depends on past)
- What if the output should depend on the whole input sequence?
- Example: in speech recognition we need to know what phonemes and words come in the future

Note

Bidirectional RNNs

- So far we have seen only RNNs with a specific temporal direction (current state depends on past)
- What if the output should depend on the whole input sequence?
- Example: in speech recognition we need to know what phonemes and words come in the future

Note

Bidirectional RNNs

- So far we have seen only RNNs with a specific temporal direction (current state depends on past)
- What if the output should depend on the whole input sequence?
- Example: in speech recognition we need to know what phonemes and words come in the future

boy bow bo bo bo

Note

Bidirectional RNNs

- So far we have seen only RNNs with a specific temporal direction (current state depends on past)
- What if the output should depend on the whole input sequence?
- Example: in speech recognition we need to know what phonemes and words come in the future

boy bow boolean bo bo

Note

Bidirectional RNNs

- So far we have seen only RNNs with a specific temporal direction (current state depends on past)
- What if the output should depend on the whole input sequence?
- Example: in speech recognition we need to know what phonemes and words come in the future

boy bow boolean bottle bo

Note

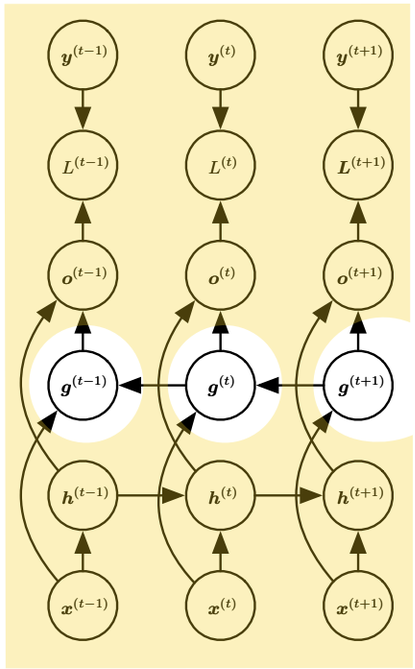
Bidirectional RNNs

- So far we have seen only RNNs with a specific temporal direction (current state depends on past)
- What if the output should depend on the whole input sequence?
- Example: in speech recognition we need to know what phonemes and words come in the future

  boy bow boolean bottle border

Note

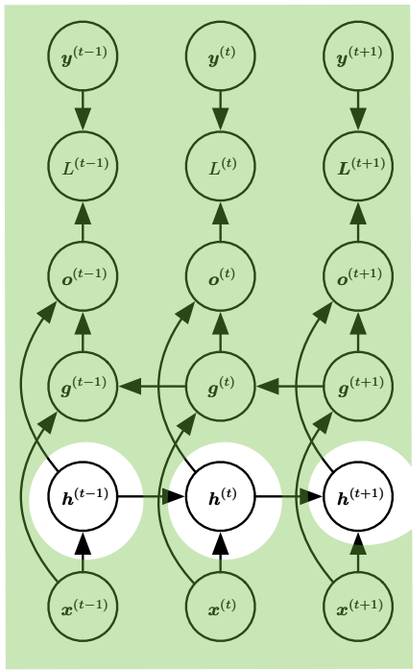
Bidirectional RNNs



causal RNN

Note

Bidirectional RNNs



anticausal RNN

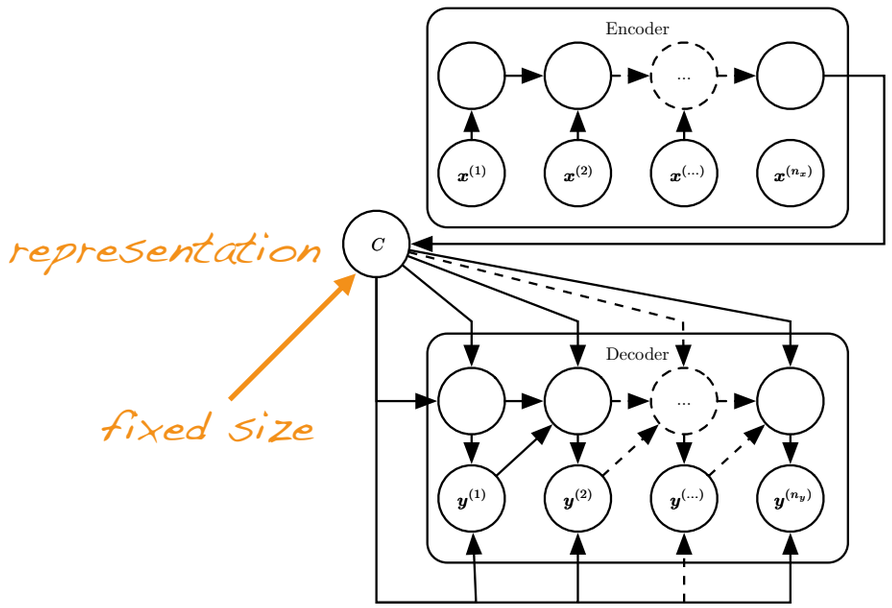
Can be extended to images by combining 4 RNNs (going in 4 directions: north, south, east and west)

Encoder-Decoder Architectures

- Mapping an input sequence to an output sequence of different length
- Applications in speech recognition, machine translation, question answering
- Input sequence is a **context**
- Build a representation of the context (a vector or a sequence of vectors)

Note

Encoder-Decoder Architectures



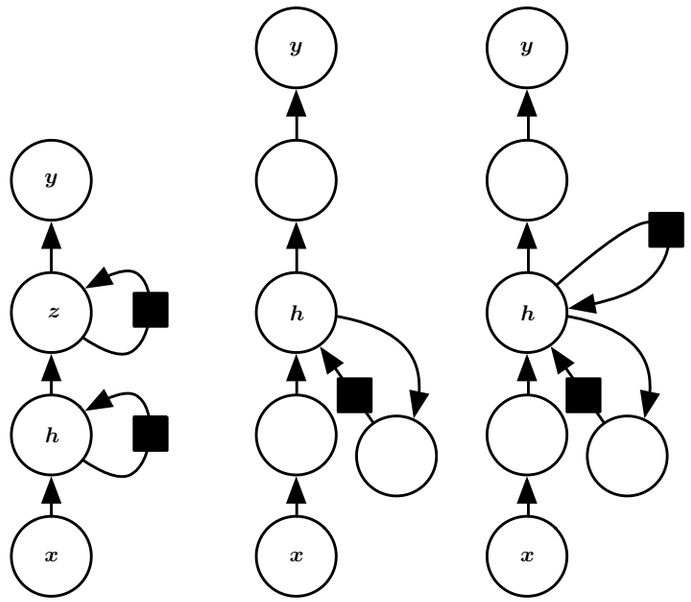
The input sequence is encoded in a single vector C and then this is fed as the same input (context) to all the nodes in a decoder network or as the initial state (or both). One potential issue is that C might be too small to represent long/complex sequences.

Deep Recurrent Networks

- Key transformations
  - from the input to the hidden state
  - between hidden states
  - from the hidden state to the output
- These are typically shallow transformations
- How about adding depth to each transformation?

Experimental evidence seems to suggest that depth helps.

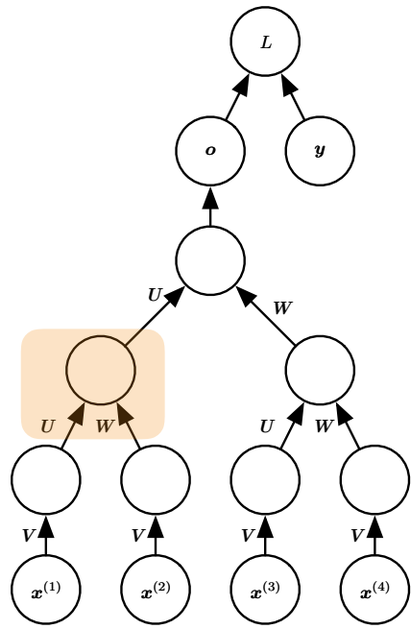
Deep Recurrent Networks



Different ways to add depth: (left) multiple recurrent units, (center) an extra layer between every transition, and (right) adding skipping connections to help the training difficulties of the center network (has very long paths).

Recursive Neural Networks

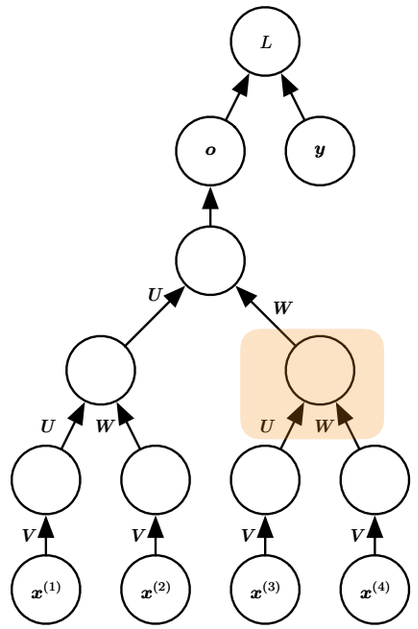
- A generalization of RNNs with a deep tree computational graph
- A sequence of length t requires a log t depth



Note

Recursive Neural Networks

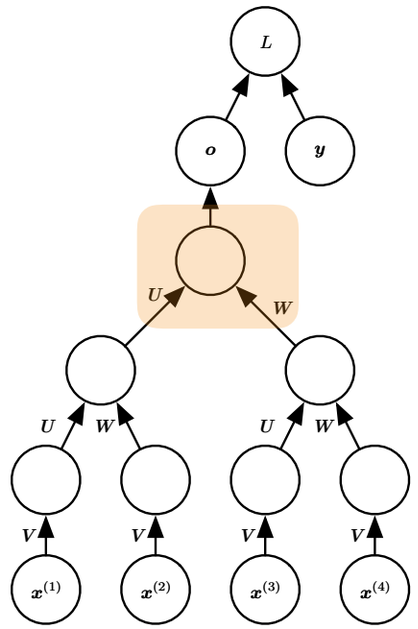
- A generalization of RNNs with a deep tree computational graph
- A sequence of length t requires a log t depth



Note

Recursive Neural Networks

- A generalization of RNNs with a deep tree computational graph
- A sequence of length t requires a log t depth



Note

Long-Term Dependencies

- Gradients propagated over many stages tend to vanish
- Consider the recurrence $h^{(t)} = W^\top h^{(t-1)}$ where $W = Q\Lambda Q^\top$
- Then, we have $h^{(t)} = (W^t)^\top h^{(0)}$

  and $h^{(t)} = Q\Lambda^t Q^\top h^{(0)}$

We see that the eigenvalues of \Lambda go quickly to 0 or to 1 or to infinity.

Long-Term Dependencies

- What does the RNN remember through

$$h^{(t)} = Q\Lambda^t Q^\top h^{(0)} \ ?$$

- Generic input $h^{(0)} = \sum_i \alpha_i q_i$ where $Q = [q_1 \ \ldots \ q_n]$
- Then, $h^{(t)} = \sum_i \alpha_i \lambda_i^t q_i$
- Coefficients corresponding to unit eigenvalues survive
- For stability to noise, eigenvalues need to be <1

Any h^(0) that is not aligned with the eigenvector corresponding to the largest eigenvalue will eventually disappear.

Long-Term Dependencies

- Long term dependencies have very small gradients compared to short term ones
- **Example**: consider learning to classify sequences into two categories (starting and ending with the same or opposite value)

The long dependencies in these two sequences will be very difficult to capture, despite the simplicity of the categories. The initial sign (1 bit of information) will be easily lost by the time the last sign is observed.

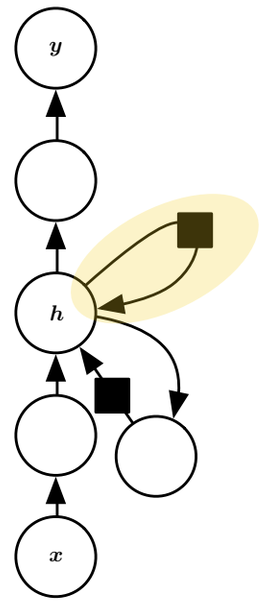
Multiple Time Scales

- Another way to handle long-term dependencies is to operate at multiple time scales
- Stategies
  - Skip connections
  - Leaky units
  - Removal of connections

In all these cases we aim at having the eigenvalues close to 1.

Skip Connections

- Use direct connections between variables at large time differences
- This reduces the rate at which gradients vanish or explode



Reduces the eigenvalues exponent by the number of skipped connections

Leaky Units

- Use linear self-connections with almost unit weight

$$\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1-\alpha)\nu^{(t-1)}$$

also called **leaky units**

This equation keeps \mu in memory for a long time if \alpha is almost 1. Vice versa, when \alpha is close to 0 the system rapidly forgets the past.

Removing Connections

- Arrange RNN state at multiple time scales
- Facilitate flow of long distances by **removing** short distance connections
- In contrast, skip connections leave a choice to the network (it can choose to focus on short or long distances)
- Different scales can be achieved via leaky units with different coefficients

In this case we encourage mostly long-term dependencies (these are the only paths we allow)

Gated RNNs

- In the family of **gated RNNs** (e.g., LSTMs, GRUs)
- Key idea is to create paths through which derivatives neither vanish nor explode
- Generalize leaky units: **learn the leak coefficient**

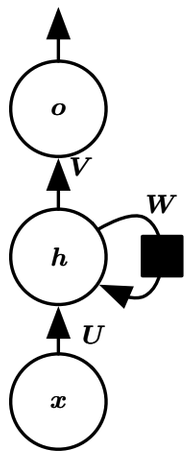
Note

Long Short-Term Memory

- Successful on handwriting recognition, speech recognition, machine translation, image captioning, image parsing
- Leaky units accumulate information over a long time
- After using the information the network might want to **forget it**
- Train the network to learn **when** to forget

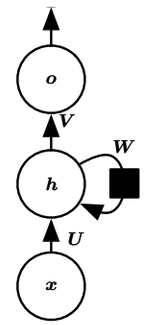
Note

Long Short-Term Memory



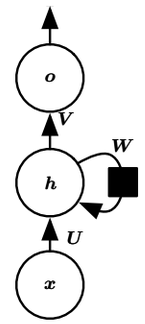
Note

Long Short-Term Memory



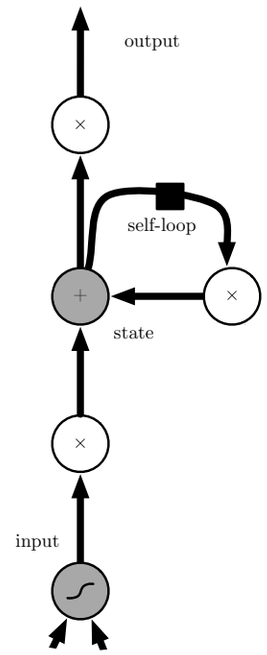
Note

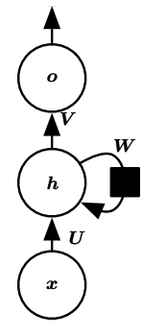
Long Short-Term Memory



We extend the RNN by introducing gating nodes between every link

Long Short-Term Memory

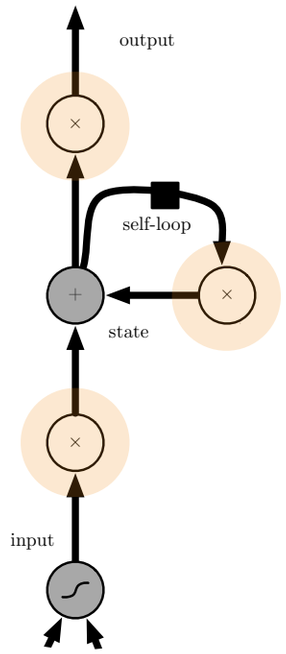


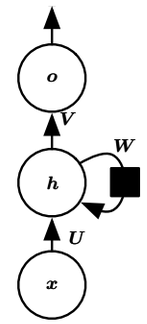


We extend the RNN by introducing gating nodes between every link

Long Short-Term Memory

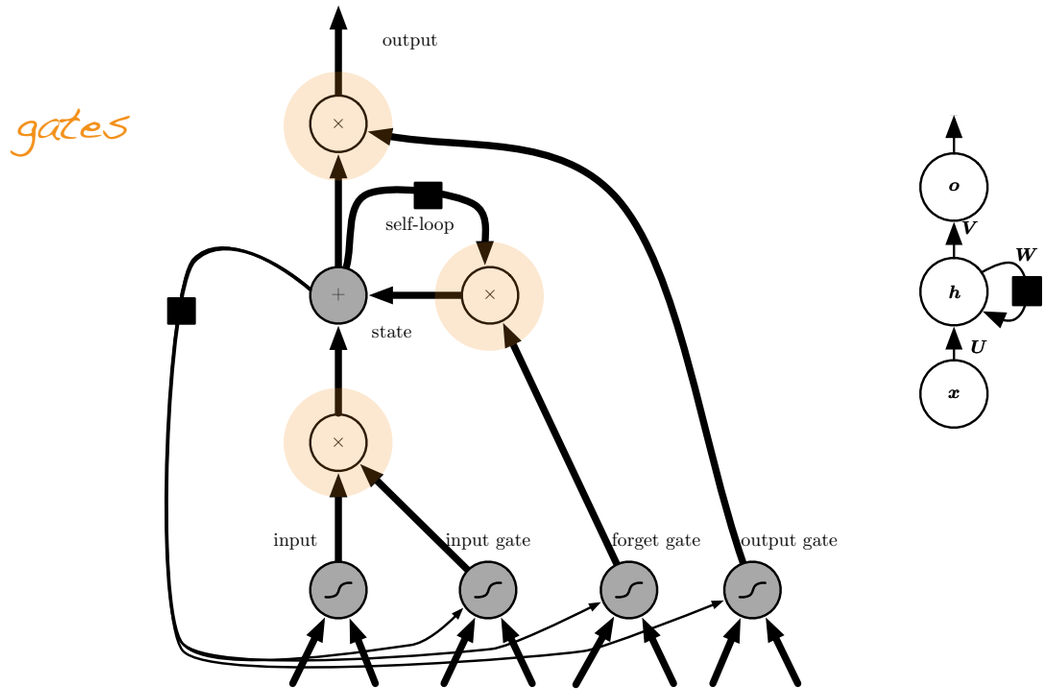
gates





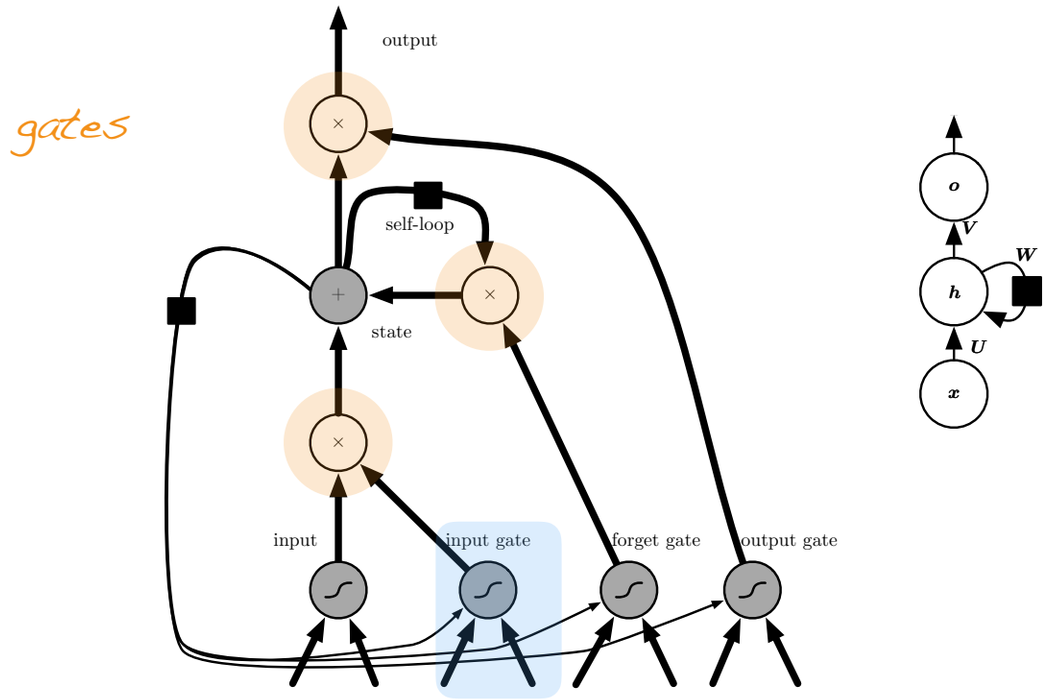
We extend the RNN by introducing gating nodes between every link

Long Short-Term Memory



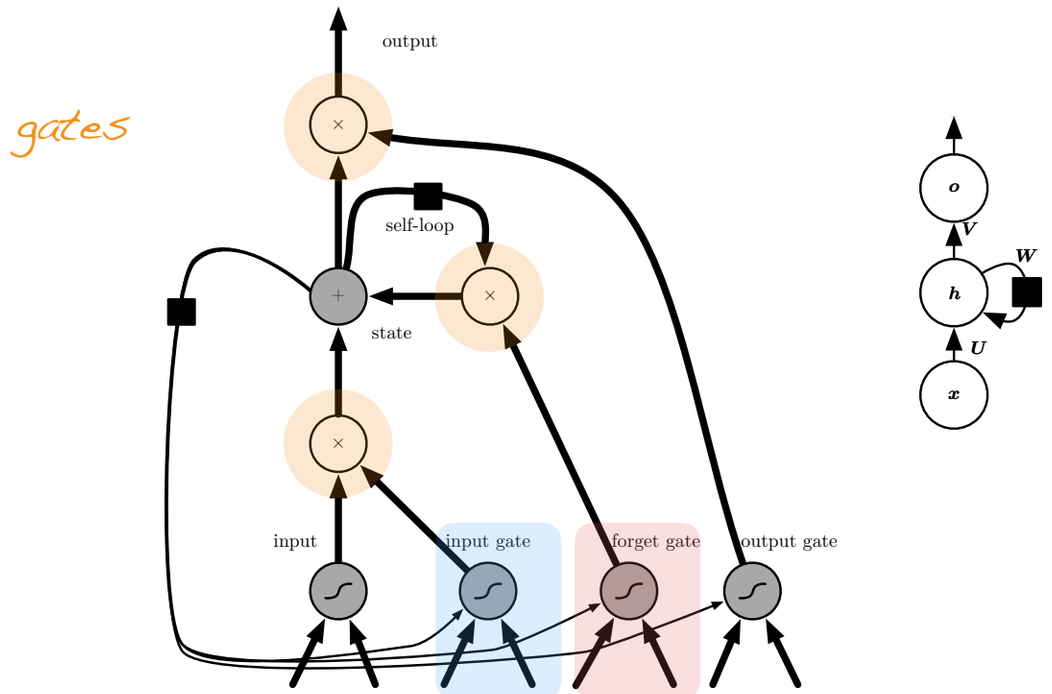
Note

Long Short-Term Memory



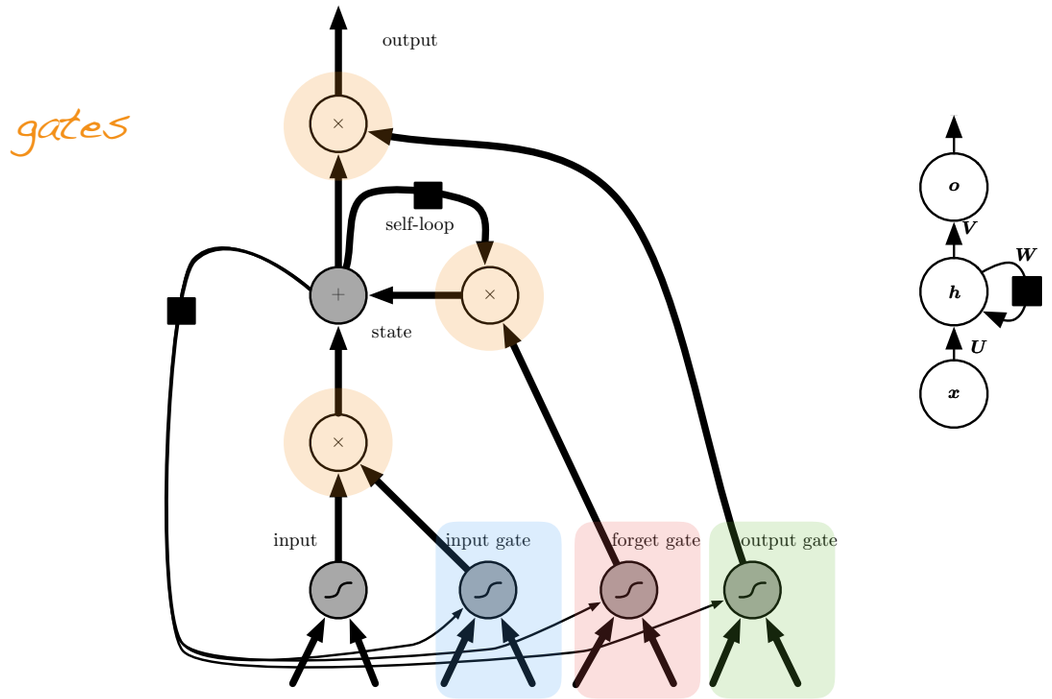
Note

Long Short-Term Memory



Note

Long Short-Term Memory



Note

Long Short-Term Memory

Dynamical System

state eq. $$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \langle \mathbf{U}_i, \mathbf{x}^{(t)} \rangle + \langle \mathbf{W}_i, \mathbf{h}^{(t-1)} \rangle\right)$$

output eq. $$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$

Gating

input gate $$g_i^{(t)} = \sigma\left(b_i^g + \langle \mathbf{U}_i^g, \mathbf{x}^{(t)} \rangle + \langle \mathbf{W}_i^g, \mathbf{h}^{(t-1)} \rangle\right)$$

state gate $$f_i^{(t)} = \sigma\left(b_i^f + \langle \mathbf{U}_i^f, \mathbf{x}^{(t)} \rangle + \langle \mathbf{W}_i^f, \mathbf{h}^{(t-1)} \rangle\right)$$

output gate $$q_i^{(t)} = \sigma\left(b_i^o + \langle \mathbf{U}_i^o, \mathbf{x}^{(t)} \rangle + \langle \mathbf{W}_i^o, \mathbf{h}^{(t-1)} \rangle\right)$$

Note

Convolutional LSTMs

Dynamical System

state eq. $$\mathbf{s}^{(t)} = \mathbf{f}^{(t)}\mathbf{s}^{(t-1)} + \mathbf{g}^{(t)}\sigma\left(\mathbf{b} + \mathbf{U} * \mathbf{x}^{(t)} + \mathbf{W} * \mathbf{h}^{(t-1)}\right)$$

output eq. $$\mathbf{h}^{(t)} = \tanh\left(\mathbf{s}^{(t)}\right)\mathbf{q}^{(t)}$$

Gating

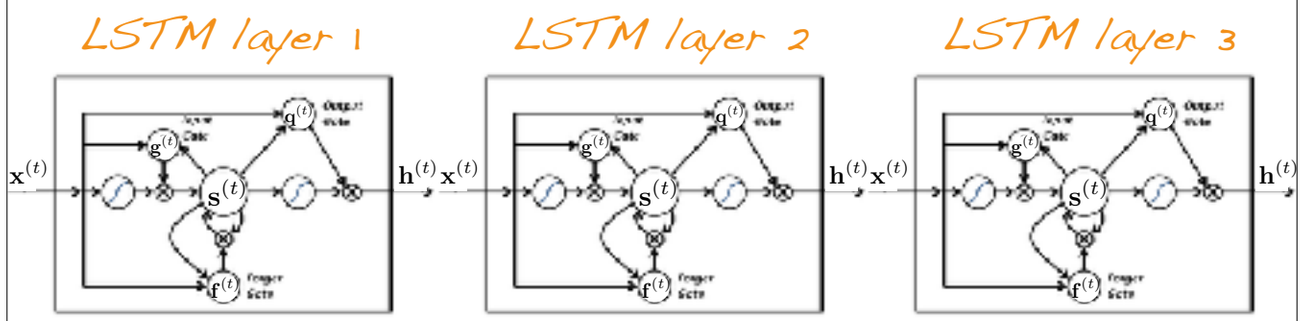
input gate $$\mathbf{g}^{(t)} = \sigma\left(\mathbf{b}^{g} + \mathbf{U}_i^{g} * \mathbf{x}^{(t)} + \mathbf{W}_i^{g} * \mathbf{h}^{(t-1)}\right)$$

state gate $$\mathbf{f}^{(t)} = \sigma\left(\mathbf{b}^{f} + \mathbf{U}_i^{f} * \mathbf{x}^{(t)} + \mathbf{W}_i^{f} * \mathbf{h}^{(t-1)}\right)$$

output gate $$\mathbf{q}^{(t)} = \sigma\left(\mathbf{b}^{o} + \mathbf{U}_i^{o} * \mathbf{x}^{(t)} + \mathbf{W}_i^{o} * \mathbf{h}^{(t-1)}\right)$$

Can deal with images and larger input data.

Stacking



Note

Gated RNNs

- The gated recurrent units (**GRUs**) is a variant
- Uses same gating for state and input (leaky unit)

output

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + \left(1 - u_i^{(t-1)}\right) \sigma\left(b_i + \langle \mathbf{U}_i, \mathbf{x}^{(t-1)} \rangle + \langle \mathbf{W}_i, \mathbf{r}^{(t-1)} \odot \mathbf{h}^{(t-1)} \rangle\right)$$

shared

gating

gating

update $\quad u_i^{(t)} = \sigma\left(b_i^u + \langle \mathbf{U}_i^u, \mathbf{x}^{(t)} \rangle + \langle \mathbf{W}_i^u, \mathbf{h}^{(t)} \rangle\right)$

reset $\quad r_i^{(t)} = \sigma\left(b_i^r + \langle \mathbf{U}_i^r, \mathbf{x}^{(t)} \rangle + \langle \mathbf{W}_i^r, \mathbf{h}^{(t)} \rangle\right)$

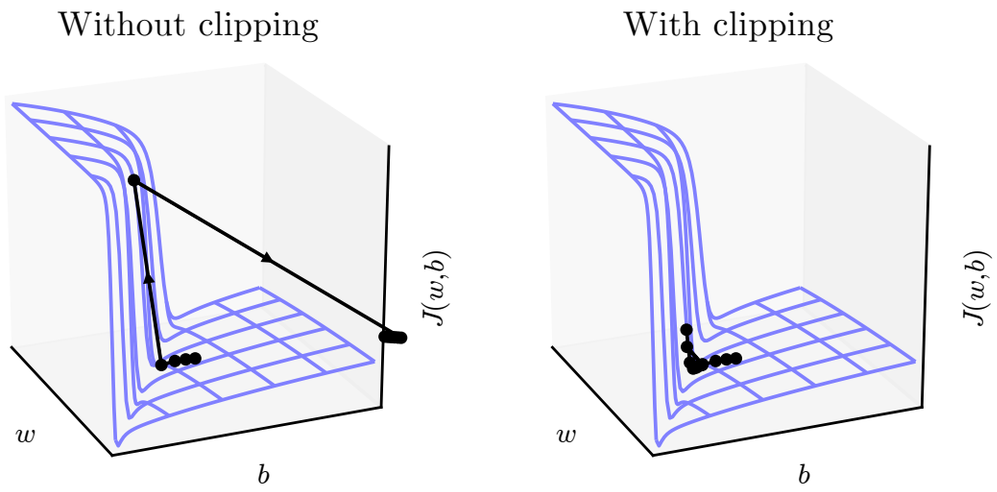
The update gate acts like a leaky unit (u=1 means copy and u=0 means forget). The reset gate controls which part of the state is used to update the next state (adds more nonlinearity to state equation).

Optimization

- Just use SGD
- Often easier to design a model easy to train than to design a more powerful optimization algorithm
- Gating introduces strong nonlinearities: Clipping gradients
- Alternative approach via weights regularization

Note

Clipping Gradients



$$\text{if } \boldsymbol{g} > v \text{ then } \boldsymbol{g} = v\frac{\boldsymbol{g}}{|\boldsymbol{g}|}$$

Using a constant learning rate (albeit slowly decaying) may cause an update trajectory as on the left. The parameters are moving within a valley when with a small update they overshoot the canyon and end up at the top of the cliff. Then the next gradient update is very large and sends the parameters far away. With gradient clipping this behavior is better controlled.

Information Flow

- Clipping helps only with exploding gradients
- To deal with vanishing gradients we introduced the LSTM gating mechanism
- In alternative one can regularize the weights so that the recurrent matrix eigenvalues are close to 1 when information should be preserved

$$\Omega = \sum_t \left| \frac{\left| (\nabla_{h^{(t)}} L) \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \right|}{|\nabla_{h^{(t)}} L|} - 1 \right|^2$$

This penalty term favors weights that preserve the gradient of L over the units. The optimisation is not straightforward and a simplification is to keep the gradient of L wrt h^t the same as it is backpropagated (basically consider only the weights in the state equation). This however does not lead to a better performance than LSTMs when enough data is available.