An Introduction to Recurrent Neural Networks

Alex Atanasov[1]

[1]Dept. of Physics
Yale University

April 24, 2018

Overview

Throughout this presentation I will be using the notation from the book by Ian Goodfellow, Yoshua Bengio, and Aaron Courville

Motivation

Two motivations for recurrent neural network models:

Motivation

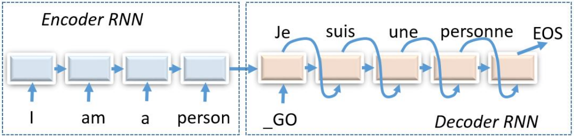
Two motivations for recurrent neural network models:

Sequential Processing

Motivation

Two motivations for recurrent neural network models:

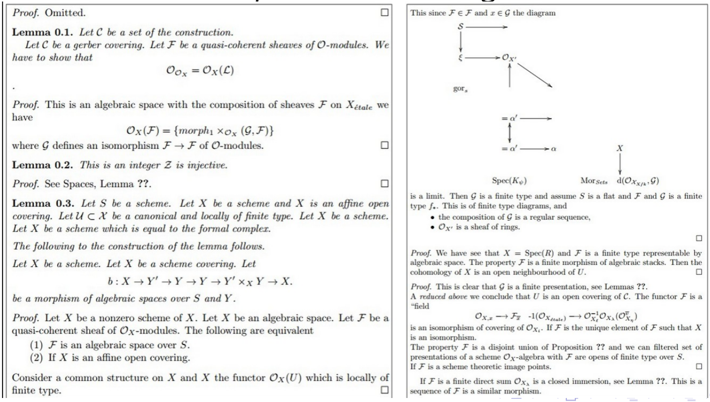
Sequential Processing



Motivation

Two motivations for recurrent neural network models:

Sequential Processing



Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.

A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \ \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.

The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.

If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

Motivation

Two motivations for recurrent neural network models:

Sequential Processing

Motivation

Two motivations for recurrent neural network models:

Modeling of Neuronal Connectivity

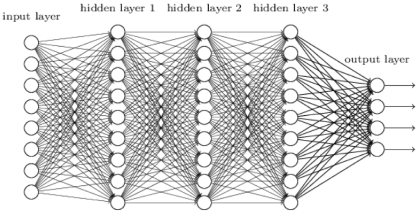
Most human brains don't look like this:

Motivation

Two motivations for recurrent neural network models:

Modeling of Neuronal Connectivity

Most human brains don't look like this:

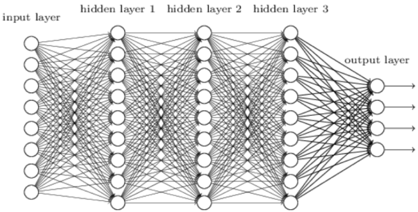


Motivation

Two motivations for recurrent neural network models:

Modeling of Neuronal Connectivity

Most human brains don't look like this[1]:



Motivation

Two motivations for recurrent neural network models:

Modeling of Neuronal Connectivity

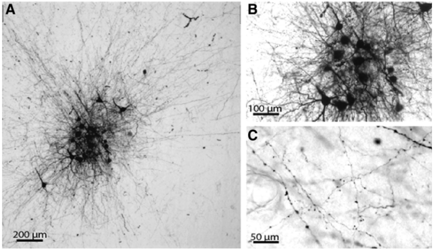
Instead, we have neurons connecting in a dense web called a *recurrent network*

Motivation

Two motivations for recurrent neural network models:

Modeling of Neuronal Connectivity

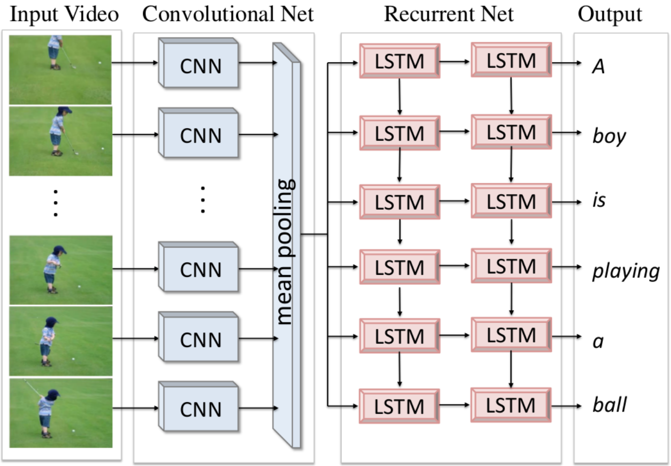
Instead, we have neurons connecting in a dense web called a *recurrent network*



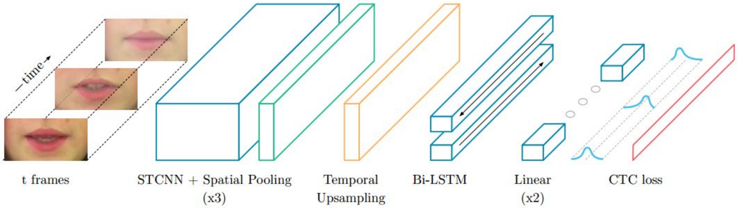
RNN Examples

We can also combine RNNs with other networks we've seen before

RNN Examples



RNN Examples



Recurrent neural networks are

Recurrent neural networks are

- Designed for processing sequential data

Recurrent neural networks are

- Designed for processing sequential data
- Like CNNs, motivated by biological example

Recurrent neural networks are

- Designed for processing sequential data
- Like CNNs, motivated by biological example
- Unlike CNNs and deep neural networks in that their neural connections can contain cycles

Recurrent neural networks are

- Designed for processing sequential data
- Like CNNs, motivated by biological example
- Unlike CNNs and deep neural networks in that their neural connections can contain cycles
- A very general form of neural network

Recurrent neural networks are

- Designed for processing sequential data
- Like CNNs, motivated by biological example
- Unlike CNNs and deep neural networks in that their neural connections can contain cycles
- A very general form of neural network
- Turing complete

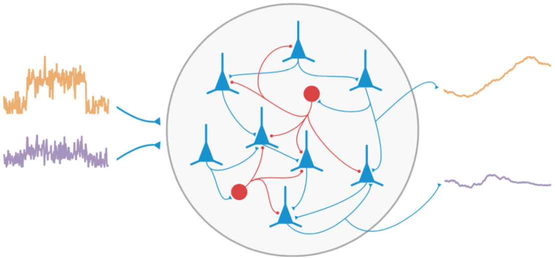
Visual Representation

From a neuroscience paper[1]:

[1] *Song et al. 2014, Training Excitatory-Inhibitory Recurrent Neural Networks for Cognitive Tasks: A Simple and Flexible Framework*

Visual Representation

From a neuroscience paper[1]:



[1] *Song et al. 2014, Training Excitatory-Inhibitory Recurrent Neural Networks for Cognitive Tasks: A Simple and Flexible Framework*

Hopfield Networks

The first formulation of a "recurrent-like" neural network was made by John Hopfield (1982)

Hopfield Networks

The first formulation of a “recurrent-like” neural network was made by John Hopfield (1982)

- Very simple form of neural network

Hopfield Networks

The first formulation of a "recurrent-like" neural network was made by John Hopfield (1982)

- Very simple form of neural network
- Build in the context of theoretical neuroscience

Hopfield Networks

The first formulation of a "recurrent-like" neural network was made by John Hopfield (1982)

- Very simple form of neural network
- Build in the context of theoretical neuroscience
- First order attempt at understanding the mechanism underlying associative memory

Hopfield Networks

The first formulation of a “recurrent-like” neural network was made by John Hopfield (1982)

- Very simple form of neural network
- Build in the context of theoretical neuroscience
- First order attempt at understanding the mechanism underlying associative memory
- Still an important object of study, primarily in neuroscience

LSTM Networks

Invented by Hochreiter and Schmidhuber in 1997

- Only recently (2007), when computational resources were powerful enough to train LSTM networks, did they begin to revolutionize the field

LSTM Networks

Invented by Hochreiter and Schmidhuber in 1997

- Only recently (2007), when computational resources were powerful enough to train LSTM networks, did they begin to revolutionize the field
- First came to fame for outperforming all traditional models in certain speech applications

LSTM Networks

Invented by Hochreiter and Schmidhuber in 1997

- Only recently (2007), when computational resources were powerful enough to train LSTM networks, did they begin to revolutionize the field
- First came to fame for outperforming all traditional models in certain speech applications
- In 2009, RNNs set records for handwriting recognition

LSTM Networks

Invented by Hochreiter and Schmidhuber in 1997

- Only recently (2007), when computational resources were powerful enough to train LSTM networks, did they begin to revolutionize the field
- First came to fame for outperforming all traditional models in certain speech applications
- In 2009, RNNs set records for handwriting recognition
- In 2015, Google's speech recognition received at 47% jump using an LSTM network

LSTM Networks

Invented by Hochreiter and Schmidhuber in 1997

- Only recently (2007), when computational resources were powerful enough to train LSTM networks, did they begin to revolutionize the field
- First came to fame for outperforming all traditional models in certain speech applications
- In 2009, RNNs set records for handwriting recognition
- In 2015, Google's speech recognition received at 47% jump using an LSTM network
- Significantly improved machine translation, language modeling and multilingual language processing (i.e. Google Translate)

LSTM Networks

Invented by Hochreiter and Schmidhuber in 1997

- Only recently (2007), when computational resources were powerful enough to train LSTM networks, did they begin to revolutionize the field
- First came to fame for outperforming all traditional models in certain speech applications
- In 2009, RNNs set records for handwriting recognition
- In 2015, Google's speech recognition received at 47% jump using an LSTM network
- Significantly improved machine translation, language modeling and multilingual language processing (i.e. Google Translate)
- Together with CNNs, significantly improved image captioning

Lets actually define what an RNN is

Lets actually define what an RNN is

Our Variables

Our input will be a sequence of vectors:

$$\mathbf{x}^{(1)} \ldots \mathbf{x}^{(\tau)}$$

Our Variables

Our input will be a sequence of vectors:

$$\mathbf{x}^{(1)} \ldots \mathbf{x}^{(\tau)}$$

Let $1 \leq t \leq \tau$ index these vectors. Our input is then denoted $\mathbf{x}^{(t)}$.

Our Variables

Our input will be a sequence of vectors:

$$\mathbf{x}^{(1)} \dots \mathbf{x}^{(\tau)}$$

Let $1 \leq t \leq \tau$ index these vectors. Our input is then denoted $\mathbf{x}^{(t)}$.

Thinking of t as "time" gives us the dynamical evolution of a system.

Our Variables

Our input will be a sequence of vectors:

$$\mathbf{x}^{(1)} \dots \mathbf{x}^{(\tau)}$$

Let $1 \leq t \leq \tau$ index these vectors. Our input is then denoted $\mathbf{x}^{(t)}$. The activity of the neurons inside the RNN at time t will denoted by the vector $\mathbf{h}^{(t)}$

Our Variables

Our input will be a sequence of vectors:

$$\mathbf{x}^{(1)} \dots \mathbf{x}^{(\tau)}$$

Let $1 \leq t \leq \tau$ index these vectors. Our input is then denoted $\mathbf{x}^{(t)}$.
The activity of the neurons inside the RNN at time t will denoted by the vector $\mathbf{h}^{(t)}$
The output at time t will be denoted $\mathbf{o}^{(t)}$

Our Variables

Our input will be a sequence of vectors:

$$\mathbf{x}^{(1)} \dots \mathbf{x}^{(\tau)}$$

Let $1 \leq t \leq \tau$ index these vectors. Our input is then denoted $\mathbf{x}^{(t)}$.
The activity of the neurons inside the RNN at time t will denoted by the vector $\mathbf{h}^{(t)}$
The output at time t will be denoted $\mathbf{o}^{(t)}$
The target output at time t will be denoted $\mathbf{y}^{(t)}$

Our Variables

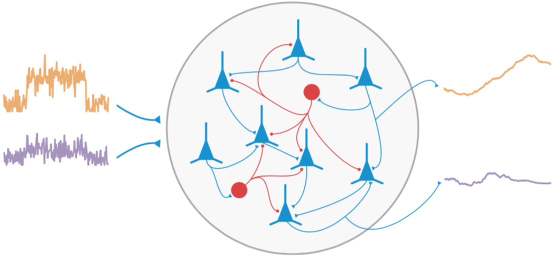
Our input will be a sequence of vectors:

$$\mathbf{x}^{(1)} \dots \mathbf{x}^{(\tau)}$$

Let $1 \leq t \leq \tau$ index these vectors. Our input is then denoted $\mathbf{x}^{(t)}$.
The activity of the neurons inside the RNN at time t will denoted by the vector $\mathbf{h}^{(t)}$
The output at time t will be denoted $\mathbf{o}^{(t)}$
The target output at time t will be denoted $\mathbf{y}^{(t)}$
The RNN's goal is to minimize a loss $L(\mathbf{o}^{(t)}, \mathbf{y}^{(t)})$ over **all times**.

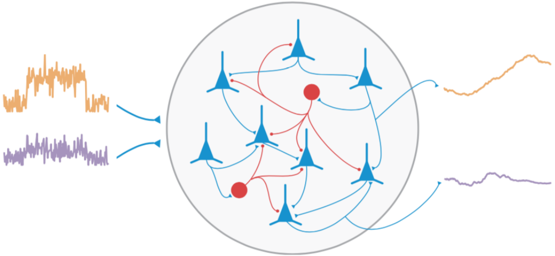
Our Variables

So:



Our Variables

So:



$\mathbf{x}^{(t)}$ $\mathbf{h}^{(t)}$ $\mathbf{o}^{(t)}$

Equations of Evolution

Given an input $\mathbf{x}^{(t)}$ with

Equations of Evolution

Given an input $\mathbf{x}^{(t)}$ with

- Input weights $\mathbf{U}_{ij}$ connecting input j to RNN neuron i

Equations of Evolution

Given an input $\mathbf{x}^{(t)}$ with

- Input weights $\mathbf{U}_{ij}$ connecting input j to RNN neuron i
- Internal weights $\mathbf{W}_{ij}$ connecting RNN neuron j to RNN neuron i and internal bias $\mathbf{b}_i$ on RNN neuron i

Equations of Evolution

Given an input $\mathbf{x}^{(t)}$ with

- Input weights $\mathbf{U}_{ij}$ connecting input j to RNN neuron i
- Internal weights $\mathbf{W}_{ij}$ connecting RNN neuron j to RNN neuron i and internal bias $\mathbf{b}_i$ on RNN neuron i
- Output weights $\mathbf{V}_{ij}$ connecting RNN neuron j to output neuron i and internal bias $\mathbf{c}_i$ on output neuron i

Equations of Evolution

Given an input $\mathbf{x}^{(t)}$ with

- Input weights $\mathbf{U}_{ij}$ connecting input j to RNN neuron i
- Internal weights $\mathbf{W}_{ij}$ connecting RNN neuron j to RNN neuron i and internal bias $\mathbf{b}_i$ on RNN neuron i
- Output weights $\mathbf{V}_{ij}$ connecting RNN neuron j to output neuron i and internal bias $\mathbf{c}_i$ on output neuron i

The time evolution of $\mathbf{h}^{(t)}, \mathbf{o}^{(t)}$ is given by:

$$\mathbf{h}^{(t)} = \tanh[\mathbf{U} \cdot \mathbf{x}^{(\mathbf{t})} + \mathbf{W} \cdot \mathbf{h}^{(\mathbf{t-1})} + \mathbf{b}]$$

Equations of Evolution

Given an input $\mathbf{x}^{(t)}$ with

- Input weights $\mathbf{U}_{ij}$ connecting input j to RNN neuron i
- Internal weights $\mathbf{W}_{ij}$ connecting RNN neuron j to RNN neuron i and internal bias $\mathbf{b}_i$ on RNN neuron i
- Output weights $\mathbf{V}_{ij}$ connecting RNN neuron j to output neuron i and internal bias $\mathbf{c}_i$ on output neuron i

The time evolution of $\mathbf{h}^{(t)}, \mathbf{o}^{(t)}$ is given by:

$$\mathbf{h}^{(t)} = \tanh[\mathbf{U} \cdot \mathbf{x}^{(\mathbf{t})} + \mathbf{W} \cdot \mathbf{h}^{(\mathbf{t-1})} + \mathbf{b}]$$

$$\mathbf{o}^{(t)} = \mathbf{V} \cdot \mathbf{h}^{(\mathbf{t})} + \mathbf{c}$$

Equations of Evolution

Given an input $\mathbf{x}^{(t)}$ with

- Input weights $\mathbf{U}_{ij}$ connecting input j to RNN neuron i
- Internal weights $\mathbf{W}_{ij}$ connecting RNN neuron j to RNN neuron i and internal bias $\mathbf{b}_i$ on RNN neuron i
- Output weights $\mathbf{V}_{ij}$ connecting RNN neuron j to output neuron i and internal bias $\mathbf{c}_i$ on output neuron i

The time evolution of $\mathbf{h}^{(t)}, \mathbf{o}^{(t)}$ is given by:

$$\mathbf{h}^{(t)} = \tanh[\mathbf{U} \cdot \mathbf{x}^{(\mathbf{t})} + \mathbf{W} \cdot \mathbf{h}^{(\mathbf{t-1})} + \mathbf{b}]$$

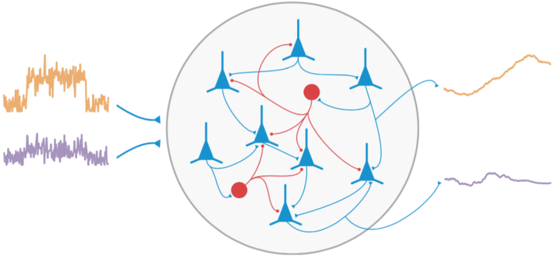
$$\mathbf{o}^{(t)} = \mathbf{V} \cdot \mathbf{h}^{(\mathbf{t})} + \mathbf{c}$$

Often, we want a probability as our output, so our RNN output is

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

Equations of Evolution

So:



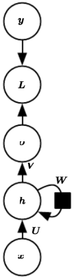
$\mathbf{x}^{(t)}$ $\mathbf{U}$ $\mathbf{h}^{(t)}, \mathbf{W}, \mathbf{b}$ $\mathbf{V}, \mathbf{c}$ $\mathbf{o}^{(t)}$

Computational Graph

Often in the field of deep learning, RNNs are pictorially described by computational graphs.

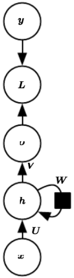
Computational Graph

Often in the field of deep learning, RNNs are pictorially described by computational graphs.



Computational Graph

Often in the field of deep learning, RNNs are pictorially described by computational graphs.



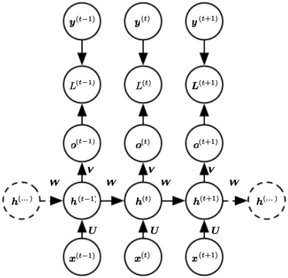
Notice this is different from the usual deep network picture

Computational Graph

To recover a deep network picture, we perform an operation known as *unrolling the graph*

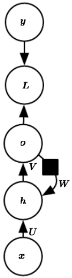
Computational Graph

To recover a deep network picture, we perform an operation known as *unrolling the graph*



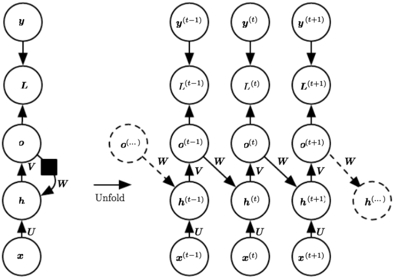
Examples of Computational Graphs

Feeding the output in:



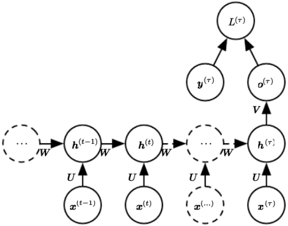
Examples of Computational Graphs

Feeding the output in:



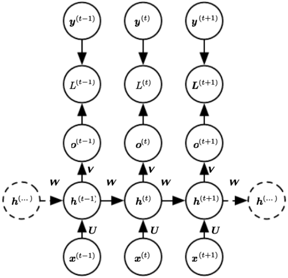
Examples of Computational Graphs

Summarizing a sequence



Gradients Descent on the Unrolled Graph

After unrolling the computational graph of an RNN, we can use the same gradient methods that we're familiar with for deep networks.



Gradients Descent on the Unrolled Graph

After unrolling the computational graph of an RNN, we can use the same gradient methods that we're familiar with for deep networks. Our trainable variables are $\mathbf{U}, \mathbf{W}, \mathbf{V}, \mathbf{b}$ and $\mathbf{c}$

Gradients Descent on the Unrolled Graph

After unrolling the computational graph of an RNN, we can use the same gradient methods that we're familiar with for deep networks. Our trainable variables are **U**, **W**, **V**, **b** and **c**

$$\nabla_{\boldsymbol{c}} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L, \tag{10.22}$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L, \tag{10.23}$$

$$\nabla_{\boldsymbol{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L)\, \boldsymbol{h}^{(t)^\top}, \tag{10.24}$$

$$\nabla_{\boldsymbol{W}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \tag{10.25}$$

$$= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)\, \boldsymbol{h}^{(t-1)^\top}, \tag{10.26}$$

$$\nabla_{\boldsymbol{U}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \tag{10.27}$$

$$= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)\, \boldsymbol{x}^{(t)^\top}, \tag{10.28}$$

Next lecture

Next lecture

- More on the the training of RNNs: vanishing and exploding gradients

Next lecture

- More on the the training of RNNs: vanishing and exploding gradients
- A tour of the many variations of RNNs

Next lecture

- More on the the training of RNNs: vanishing and exploding gradients
- A tour of the many variations of RNNs
  - Hopfield

Next lecture

- More on the the training of RNNs: vanishing and exploding gradients
- A tour of the many variations of RNNs
  - Hopfield
  - LSTM

Next lecture

- More on the the training of RNNs: vanishing and exploding gradients
- A tour of the many variations of RNNs
  - Hopfield
  - LSTM
  - Neural Turing Machines