An Introduction to Recurrent Neural Networks, Part 2

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

Review

Recurrent neural networks are

Review

Recurrent neural networks are

- Designed for processing sequential data

Review

Recurrent neural networks are

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

Review

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

Review

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

Review

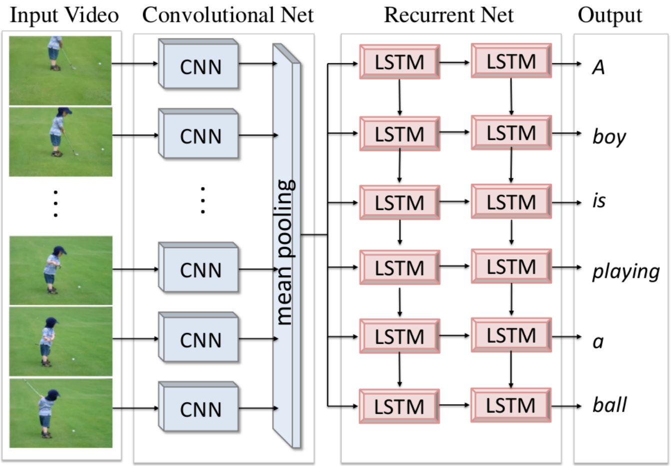
Recurrent neural networks are

- 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

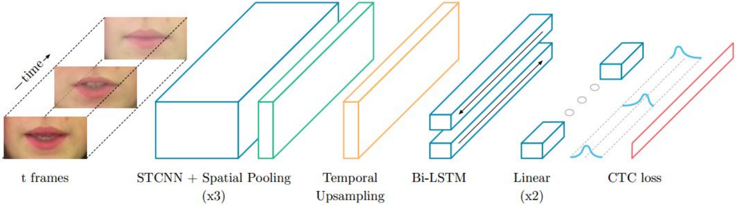
Review

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

Review

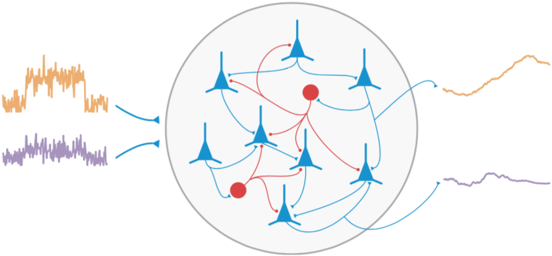

Input Video
Convolutional Net
Recurrent Net
Output
CNN
CNN
CNN
CNN
CNN
mean pooling
LSTM
A
boy
is
playing
a
ball

Review


time
t frames
STCNN + Spatial Pooling
(x3)
Temporal
Upsampling
Bi-LSTM
Linear
(x2)
CTC loss

Review

Depiction in neuroscience:



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:

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

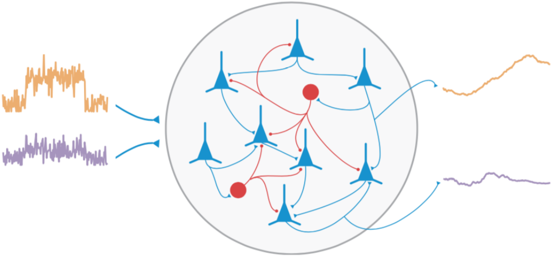
$$\mathbf{o}^{(t)} = \mathbf{V} \cdot \mathbf{h}^{(\mathbf{t})} + \mathbf{c}$$

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:



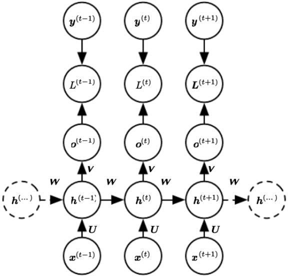
$\mathbf{x}^{(t)}$ $\quad$ $\mathbf{U}$ $\quad$ $\mathbf{h}^{(t)}$, $\mathbf{W}$, $\mathbf{b}$ $\quad$ $\mathbf{V}$, $\mathbf{c}$ $\quad$ $\mathbf{o}^{(t)}$

Review

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

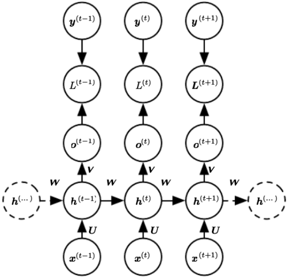
Review

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



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**

$$\begin{aligned}
\nabla_{\boldsymbol{c}} L &= \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L, && (10.22)\\
\nabla_{\boldsymbol{b}} L &= \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1-\left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L, && (10.23)\\
\nabla_{\boldsymbol{V}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L)\, \boldsymbol{h}^{(t)^\top}, && (10.24)\\
\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} && (10.25)\\
&= \sum_t \operatorname{diag}\left(1-\left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)\, \boldsymbol{h}^{(t-1)^\top}, && (10.26)\\
\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} && (10.27)\\
&= \sum_t \operatorname{diag}\left(1-\left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)\, \boldsymbol{x}^{(t)^\top}, && (10.28)
\end{aligned}$$

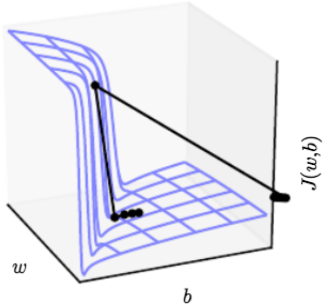
Motivation for Vanishing/Exploding gradients

The functions computed by deep neural networks and RNNs are strongly nonlinear in general.

Motivation for Vanishing/Exploding gradients

The functions computed by deep neural networks and RNNs are strongly nonlinear in general.
Consequently, they tend to have derivatives that can be either very large or very small in magnitude

Exploding Gradient



To rectify this, we employ a technique known as *clipping the gradient*.

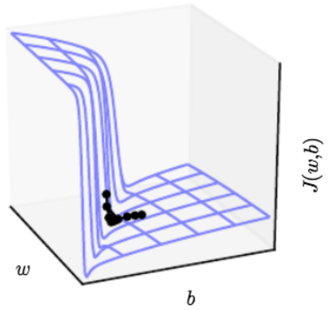
To rectify this, we employ a technique known as *clipping the gradient*.
This can refer to one of several different approaches to minimize the possibility of having large/vanishing gradients affect training.

To rectify this, we employ a technique known as *clipping the gradient*.
This can refer to one of several different approaches to minimize the possibility of having large/vanishing gradients affect training.

Example:

$$\text{if } ||\mathbf{g}|| > v, \qquad \mathbf{g} \leftarrow \frac{\mathbf{g}v}{||\mathbf{g}||}.$$

Clipped Gradient



Long Term Dependencies

Gradient clipping handles exploding gradients, but what about gradients that become increasingly smaller?

Long Term Dependencies

Gradient clipping handles exploding gradients, but what about gradients that become increasingly smaller?
The idea is that we would like $\nabla_{\mathbf{h}^{(t)}} L$ to be of roughly the same magnitude in time. This is seen as encouraging "information flow".

Long Term Dependencies

Gradient clipping handles exploding gradients, but what about gradients that become increasingly smaller?

The idea is that we would like $\nabla_{\mathbf{h}^{(t)}} L$ to be of roughly the same magnitude in time. This is seen as encouraging "information flow".

Another way to say this is we want:

$$|\nabla_{\mathbf{h}^{(t-1)}} L| = |(\nabla_{\mathbf{h}^{(t)}} L) \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}}| \approx |\nabla_{\mathbf{h}^{(t)}} L|$$

Long Term Dependencies

Gradient clipping handles exploding gradients, but what about gradients that become increasingly smaller?
The idea is that we would like $\nabla_{\mathbf{h}^{(t)}} L$ to be of roughly the same magnitude in time. This is seen as encouraging "information flow". Another way to say this is we want:

$$|\nabla_{\mathbf{h}^{(t-1)}} L| = |(\nabla_{\mathbf{h}^{(t)}} L) \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}}| \approx |\nabla_{\mathbf{h}^{(t)}} L|$$

With this motivation, *Pascanu* came up with a regularizer to constrain the parameters of a given RNN:

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

Combined with gradient clipping, this regularizer can considerably increase the span of the dependencies that an RNN can learn.

With this motivation, *Pascanu* came up with a regularizer to constrain the parameters of a given RNN:

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

Combined with gradient clipping, this regularizer can considerably increase the span of the dependencies that an RNN can learn. This is still not as effective as LSTMs in cases where training data is abundant.

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

Hopfield Networks: Mathematical Description

Take G a *complete, undirected* graph with vertex set V and edge weights w_{ij} such that

Hopfield Networks: Mathematical Description

Take G a *complete, undirected* graph with vertex set V and edge weights w_{ij} such that

- Symmetry: $w_{ij} = w_{ji}$

Hopfield Networks: Mathematical Description

Take G a *complete, undirected* graph with vertex set V and edge weights w_{ij} such that

- Symmetry: $w_{ij} = w_{ji}$
- No autapse: $w_{ii} = 0$

Hopfield Networks: Mathematical Description

Take G a *complete, undirected* graph with vertex set V and edge weights w_{ij} such that

- Symmetry: $w_{ij} = w_{ji}$
- No autapse: $w_{ii} = 0$

The activity of each neuron $v \in V$ is either -1 or 1. Further, each neuron has a threshold θ_i which determines how its activity is updated, by:

Hopfield Networks: Mathematical Description

Take G a *complete, undirected* graph with vertex set V and edge weights w_{ij} such that

- Symmetry: $w_{ij} = w_{ji}$
- No autapse: $w_{ii} = 0$

The activity of each neuron $v \in V$ is either -1 or 1. Further, each neuron has a threshold θ_i which determines how its activity is updated, by:

$$s_i^{(t+1)} \leftarrow \begin{cases} +1 \text{ if } \sum_j w_{ij} s_j^{(t)} > \theta_i \\ -1 \text{ otherwise} \end{cases}$$

Hopfield Networks: Mathematical Description

Hopfield networks can be updated

Hopfield Networks: Mathematical Description

Hopfield networks can be updated

- Synchronously: All neurons update at each timestep

Hopfield Networks: Mathematical Description

Hopfield networks can be updated

- Synchronously: All neurons update at each timestep
- Asynchronously: At each timestep a single (usually random) neuron is updated

Hopfield Networks: Mathematical Description

A Hopfield network has a given *energy function* associated to it

Hopfield Networks: Mathematical Description

A Hopfield network has a given *energy function* associated to it

$$E = -\frac{1}{2}\sum_{ij} w_{ij} s_i s_j - \sum_i \theta_i s_i$$

Hopfield Networks: Mathematical Description

A Hopfield network has a given *energy function* associated to it

$$E = -\frac{1}{2}\sum_{ij} w_{ij} s_i s_j - \sum_i \theta_i s_i$$

For physicists this might be recognized as the energy function of the *Ising model*

Hopfield Networks: Mathematical Description

A Hopfield network has a given *energy function* associated to it

$$E = -\frac{1}{2}\sum_{ij} w_{ij} s_i s_j - \sum_i \theta_i s_i$$

For physicists this might be recognized as the energy function of the *Ising model*

Given a set of memory vectors $\{\mathbf{v}_i\}_{i=1}^n$ Hopfield network weights are trained so that the energy function is close to a minimum when $\mathbf{s}$ takes the form of one of the $\mathbf{v}_i$.

Hopfield Networks: Mathematical Description

A Hopfield network has a given *energy function* associated to it

$$E = -\frac{1}{2}\sum_{ij} w_{ij} s_i s_j - \sum_i \theta_i s_i$$

For physicists this might be recognized as the energy function of the *Ising model*

Given a set of memory vectors $\{\mathbf{v}_i\}_{i=1}^n$ Hopfield network weights are trained so that the energy function is close to a minimum when $\mathbf{s}$ takes the form of one of the $\mathbf{v}_i$.

This gives a model for associative memory in the human brain, and can be used to store many states of information in a single 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

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)
- Used for *Siri, Alexa, Apple's predictive typing*, and many more

LSTM Motivation

- Learning over long-term dependences of a dataset is a challenge in deep learning.

LSTM Motivation

- Learning over long-term dependences of a dataset is a challenge in deep learning.
- This can be referred to as *fixing the time scale of integration* for neurons networks more generally

LSTM Motivation

- Learning over long-term dependences of a dataset is a challenge in deep learning.
- This can be referred to as *fixing the time scale of integration* for neurons networks more generally
- Often causes vanishing gradients and (occasionally) exploding gradients to occur

LSTM Motivation

- Learning over long-term dependences of a dataset is a challenge in deep learning.
- This can be referred to as *fixing the time scale of integration* for neurons networks more generally
- Often causes vanishing gradients and (occasionally) exploding gradients to occur
- Perhaps there is a way to modify the individual neurons to make them more amenable to holding memory for longer periods?

LSTM Motivation

- Instead of a unit that simply applies an element-wise nonlinearity to the affine transformation of inputs and recurrent units, LSTM "cells" have an internal self-loop, in addition to the outer self-loops of the RNN.

LSTM Motivation

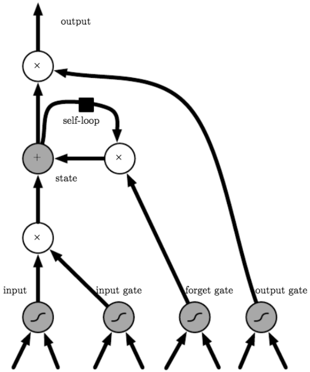
- Instead of a unit that simply applies an element-wise nonlinearity to the affine transformation of inputs and recurrent units, LSTM "cells" have an internal self-loop, in addition to the outer self-loops of the RNN.
- By making the weight of this internal self-loop gated (controlled by another hidden unit), the time scale of integration can be changed dynamically.

LSTM Motivation

- Instead of a unit that simply applies an element-wise nonlinearity to the affine transformation of inputs and recurrent units, LSTM "cells" have an internal self-loop, in addition to the outer self-loops of the RNN.
- By making the weight of this internal self-loop gated (controlled by another hidden unit), the time scale of integration can be changed dynamically.
- This is controlled by a "forget gate", illustrated below:

LSTM Networks

A single LSTM "cell", to replace a unit neuron of a standard RNN:



LSTM Networks

More explicitly, the input, forget, and output gates control the "effective weight" for the input weight, self-connection, and output weight respectively.

LSTM Networks

More explicitly, the input, forget, and output gates control the "effective weight" for the input weight, self-connection, and output weight respectively.

The internal "state" s_i of neuron i is updated as:

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

LSTM Networks

More explicitly, the input, forget, and output gates control the "effective weight" for the input weight, self-connection, and output weight respectively.
The internal "state" s_i of neuron i is updated as:

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

and its output is given by:

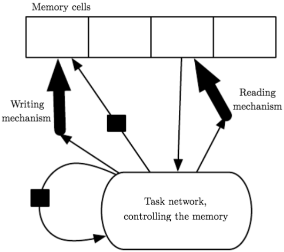
$$h_i^{(t)} = \underline{q_i^{(t)}} \tanh\left(s_i^{(t)}\right)$$

Neural Turing Machines

By allowing neural networks to have read-write access from an external memory source, they can perform operations that regular RNNs are not naively able to perform.

Neural Turing Machines

By allowing neural networks to have read-write access from an external memory source, they can perform operations that regular RNNs are not naively able to perform.



Conclusion

Recurrent Neural Networks constitute a wide variety of models of interest across the fields of deep learning, neuroscience, and even physics

Conclusion

Recurrent Neural Networks constitute a wide variety of models of interest across the fields of deep learning, neuroscience, and even physics

- RNNs are powerful tools for processing sequential data containing context and long-term dependencies

Conclusion

Recurrent Neural Networks constitute a wide variety of models of interest across the fields of deep learning, neuroscience, and even physics

- RNNs are powerful tools for processing sequential data containing context and long-term dependencies
- Their training involves vanishing/exploding gradients that need to be carefully handled

Conclusion

Recurrent Neural Networks constitute a wide variety of models of interest across the fields of deep learning, neuroscience, and even physics

- RNNs are powerful tools for processing sequential data containing context and long-term dependencies
- Their training involves vanishing/exploding gradients that need to be carefully handled
- LSTMs and other gated-type architectures are particularly powerful in being able to promote information flow and develop short-term memory