Sequence Modeling: Recurrent Neural Networks

High Dimensional and Deep Learning

INSA Toulouse – Applied Mathematics, 5th year

Juliette Chevallier, Office 109
juliette.chevallier@insa-toulouse.fr

Sequence Modeling

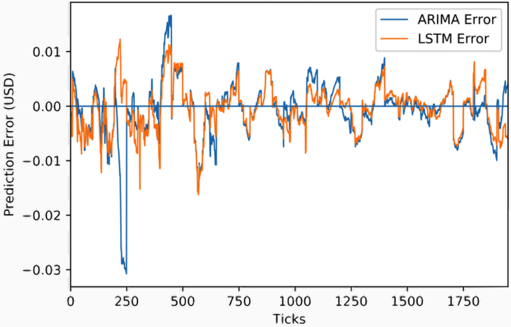
Sequential Data

Sequence $\leftrightarrow$ Explicit order on the observations that must be preserved when training models and making predictions.

- **Sequence *Prediction***: Weather forecasting, Stock market prediction, Product recommendation
- **Sequence *Classification***: DNA seq. classification, Anomaly detection, Sentiment analysis;
- **Sequence *Generation***: Text generation, Handwriting prediction, Music generation;
- **Sequence-to-Sequence *Prediction***: Multi-Step time series forecasting, Text summarization, Program execution.

From: Vinyals, Toshev, Bengio, Erhan. Show and tell: A neural image caption generator. *CVPR* 2015

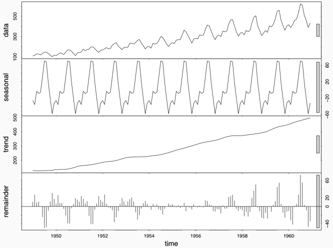
Don't be too quick to throw away traditional models !



Baughman, Haas, Wolski, Foster, Chard. *Predicting Amazon Spot Prices with LSTM Networks*. 2018.

Sequence Modeling

Time Series Forecasting



AR	Autoregressive
MA	Moving Average
ARMA	Autoregressive Moving Average
ARIMA	Autoregressive Integrated Moving Average
SARIMA	Seasonal Autoregressive Integrated Moving Average

- Describing temporal dynamics in great detail;
- Realization of a stochastic process;
- Decomposition: trend, seasonality and (stochastic) reminder;

ARIMA *vs.* (Recurrent) Neural Network

ARIMA:

+ Out-perform deep learning methods for forecasting **short-term**
+ Out-perform deep learning methods for forecasting on **univariate** data

Neural Network:

ARIMA *vs.* (Recurrent) Neural Network

ARIMA:

\+ Out-perform deep learning methods for forecasting **short-term**

\+ Out-perform deep learning methods for forecasting on **univariate** data

– Focus on univariate data, with linear relationships, and fixed and manually-diagnosed temporal dependence.

Neural Network:

ARIMA *vs.* (Recurrent) Neural Network

ARIMA:

+ Out-perform deep learning methods for forecasting **short-term**
+ Out-perform deep learning methods for forecasting on **univariate** data

– Focus on univariate data,
with linear relationships,
and fixed and manually-diagnosed
temporal dependence.

Neural Network:

+ Ability to learn noisy and non-linear relationships, regardless of the number of inputs
+ Relax univariate assumption

ARIMA *vs.* (Recurrent) Neural Network

ARIMA:

+ Out-perform deep learning methods for forecasting **short-term**
+ Out-perform deep learning methods for forecasting on **univariate** data

– Focus on univariate data,
with linear relationships,
and fixed and manually-diagnosed
temporal dependence.

Neural Network:

+ Ability to learn noisy and non-linear relationships, regardless of the number of inputs
+ Relax univariate assumption

– More complicated and difficult to train
– Do not exceed the performance of ARIMA in most cases

ARIMA *vs.* (Recurrent) Neural Network

ARIMA:

+ Out-perform deep learning methods for forecasting **short-term**
+ Out-perform deep learning methods for forecasting on **univariate** data

− Focus on univariate data, with linear relationships, and fixed and manually-diagnosed temporal dependence.

Neural Network:

+ Ability to learn noisy and non-linear relationships, regardless of the number of inputs
+ Relax univariate assumption

− More complicated and difficult to train
− Do not exceed the performance of ARIMA in most cases

$\leadsto$ **Neural Networks**

Naive approaches

Sequential Data: Audio, Speech, Language, Videos with time context, Time series

Question: *How can we integrate this context in neural networks?*

Sequential Data: Audio, Speech, Language, Videos with time context, Time series

Question*: How can we integrate this context in neural networks?*

Multilayer Perceptron Regression:

Time series prediction $\longleftrightarrow$ Regression problem: x_{t+1} as a function of x_t

$\rightsquigarrow$ *Multilayer Perceptron model*

- Difficult training,
- No more efficient than an ARIMA model (or even less)

Naive approaches

Sequential Data: Audio, Speech, Language, Videos with time context, Time series

Question: *How can we integrate this context in neural networks?*

Multilayer Perceptron Regression:

Time series prediction $\longleftrightarrow$ Regression problem: x_{t+1} as a function of x_t

$\rightsquigarrow$ *Multilayer Perceptron model*

- Difficult training,
- No more efficient than an ARIMA model (or even less)

Huge Convolutional Network:

Feed the whole sequence to a huge network

- Inefficient memory usage,
- Difficult/Impossible to train,
- Difference between spatial and temporal dimensions?
- Not real-time (translation !)

Do not do that!

Sequential Data: Audio, Speech, Language, Videos with time context, Time series

Question: *How can we integrate this context in neural networks?*

Multilayer Perceptron Regression:

Time series prediction $\longleftrightarrow$ Regression problem: x_{t+1} as a function of x_t

$\rightsquigarrow$ *Multilayer Perceptron model*

- Difficult training,
- No more efficient than an ARIMA model (or even less)

Huge Convolutional Network:

Feed the whole sequence to a huge network

- Inefficient memory usage,
- Difficult/Impossible to train,
- Difference between spatial and temporal dimensions?
- Not real-time (translation !)

Do not do that!

$\rightsquigarrow$ **Recurrent Neural Networks**

Outline

Recurrent Neural Networks

Information Persistence – Parameter-Sharing

- Humans don't start their thinking from scratch every second:
  While reading, each word is understood according to your previous ones.
  Your thoughts have persistence.

- Traditional neural networks can't do this.

Information Persistence – Parameter-Sharing

- Humans don't start their thinking from scratch every second:
  While reading, each word is understood according to your previous ones.
  Your thoughts have persistence.

- Traditional neural networks can't do this.

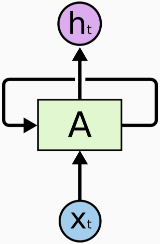
$\rightsquigarrow$ **Recurrent Neural Networks:**
Networks with loops in them, allowing information to persist.

Information Persistence – Parameter-Sharing

- Humans don't start their thinking from scratch every second:
  While reading, each word is understood according to your previous ones.
  Your thoughts have persistence.

- Traditional neural networks can't do this.

⇝ **Recurrent Neural Networks:**
Networks with loops in them, allowing information to persist.



`# Input` x_t at time t

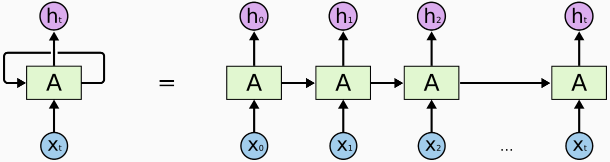
`# Output` h_t at time t

`# Loop` allows information to be passed from one step of the network to the next

Little (1974), Hopfield (1982), Rumelhart, Hinton & Williams (1986), Elman (1990)

Unfolding Computational Graphs

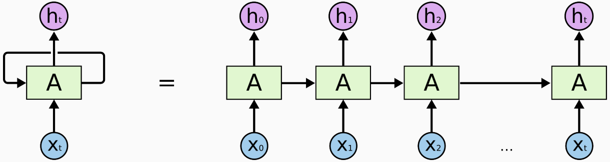
- RNN $\equiv$ Multiple copies of the **same** network,
  each passing a message to its successor: $h_t = \boldsymbol{f}(h_{t-1}, x_t; \theta)$.



- **Advantages of the unfoldingprocess:**
  - Whatever the sequence length, the learned model always has the same input size,
  - Possible to use the **same** transition f with same parameters at every time step.

Unfolding Computational Graphs

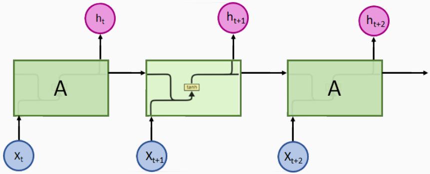
- RNN $\equiv$ Multiple copies of the **same** network,
  each passing a message to its successor: $h_t = \boldsymbol{f}(h_{t-1}, x_t; \theta)$.



- **Advantages of the unfoldingprocess:**
  - Whatever the sequence length, the learned model always has the same input size,
  - Possible to use the **same** transition f with same parameters at every time step.

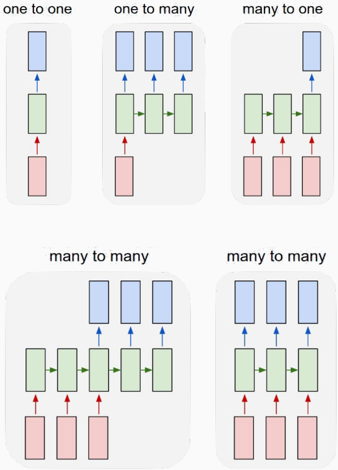
$\leadsto$ *A single model f that operates*

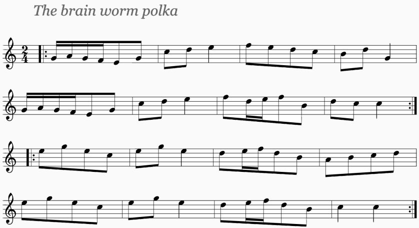
- *on all time steps,*
- *and all sequence lengths.*



Different types of RNN's:

- One-to-one *e.g.* Image classification,
- One-to-Many *e.g.* Image captioning,
- Many-to-One *e.g.* Sentiment analysis,
- Many-to-Many *e.g.* Machine Translation;





themachinefolksession.org

RNN:

1. combine the input vector
2. with their state vector
3. using a fixed (but learned) function
4. to produce a new state vector

RNNs are Turing-Complete

RNN:

1. combine the input vector
2. with their state vector
3. using a fixed (but learned) function
4. to produce a new state vector

Programming terms: Running a fixed program with certain inputs and some internal variables

RNNs essentially describe programs

RNNs are Turing-Complete

RNN:

1. combine the input vector
2. with their state vector
3. using a fixed (but learned) function
4. to produce a new state vector

Programming terms: Running a fixed program with certain inputs and some internal variables

RNNs essentially describe programs

RNNs are Turing-Complete

They can simulate arbitrary programs (with proper weights)

- Similar to universal approximation theorems for neural nets
- Shouldn't read too much into this.

Siegelmann and Sontag (1992)

Recurrent Neural Networks

Bidirectional RNNs

- All the recurrent networks we have considered so far have a causal structure:

  State at time t captures only information from the past $x_1, \ldots, x_{t-1}$, and the present input x_t

- In many *applications*, however, we want to output a prediction of y_t that may depend on the whole input sequence

  Handwriting recognition, Speech recognition, Bio-informatics, *etc.*

Bidirectional RNNs

- All the recurrent networks we have considered so far have a causal structure:

  State at time t captures only information from the past $x_1, \ldots, x_{t-1}$, and the present input x_t

- In many *applications*, however, we want to output a prediction of y_t that may depend on the whole input sequence

  Handwriting recognition, Speech recognition, Bio-informatics, *etc.*

$\rightsquigarrow$ **Bidirectional recurrent neural networks**

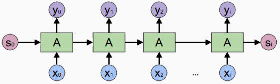
- All the recurrent networks we have considered so far have a causal structure:

    State at time t captures only information from the past $x_1, \ldots, x_{t-1}$, and the present input x_t

- In many *applications*, however, we want to output a prediction of y_t that may depend on the whole input sequence

    Handwriting recognition, Speech recognition, Bio-informatics, *etc.*

$\rightsquigarrow$ **Bidirectional recurrent neural networks**



Schuster & Paliwal (1997), Graves (2012)

Bidirectional RNNs

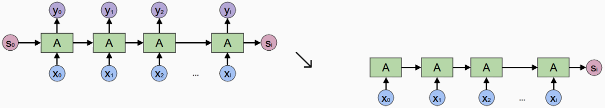
- All the recurrent networks we have considered so far have a causal structure:

  State at time t captures only information from the past $x_1, \ldots, x_{t-1}$, and the present input x_t

- In many *applications*, however, we want to output a prediction of y_t that may depend on the whole input sequence

  Handwriting recognition, Speech recognition, Bio-informatics, *etc.*

$\rightsquigarrow$ **Bidirectional recurrent neural networks**



Schuster & Paliwal (1997), Graves (2012)

Bidirectional RNNs

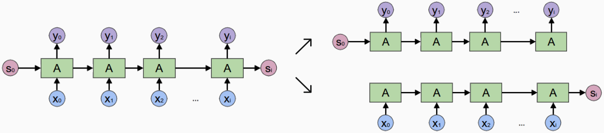
- All the recurrent networks we have considered so far have a causal structure:

  State at time t captures only information from the past $x_1, \ldots, x_{t-1}$, and the present input x_t

- In many *applications*, however, we want to output a prediction of y_t that may depend on the whole input sequence

  Handwriting recognition, Speech recognition, Bio-informatics, *etc.*

$\leadsto$ **Bidirectional recurrent neural networks**



Schuster & Paliwal (1997), Graves (2012)

Bidirectional RNNs

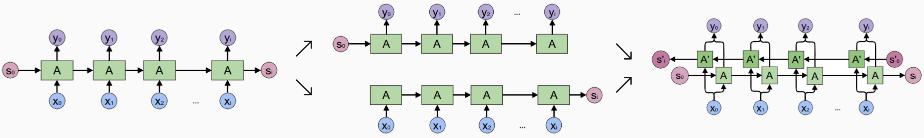
- All the recurrent networks we have considered so far have a causal structure:

  State at time t captures only information from the past $x_1, \ldots, x_{t-1}$, and the present input x_t

- In many *applications*, however, we want to output a prediction of y_t that may depend on the whole input sequence

  Handwriting recognition, Speech recognition, Bio-informatics, *etc.*

$\rightsquigarrow$ **Bidirectional recurrent neural networks**



Schuster & Paliwal (1997), Graves (2012)

Bidirectional RNNs

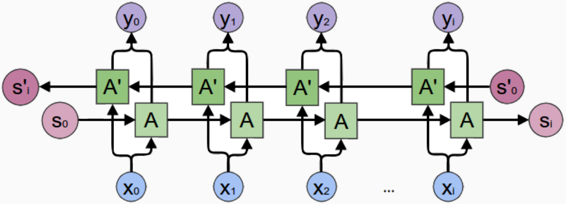
- All the recurrent networks we have considered so far have a causal structure:

  State at time t captures only information from the past $x_1, \ldots, x_{t-1}$, and the present input x_t

- In many *applications*, however, we want to output a prediction of y_t that may depend on the whole input sequence

  Handwriting recognition, Speech recognition, Bio-informatics, *etc.*

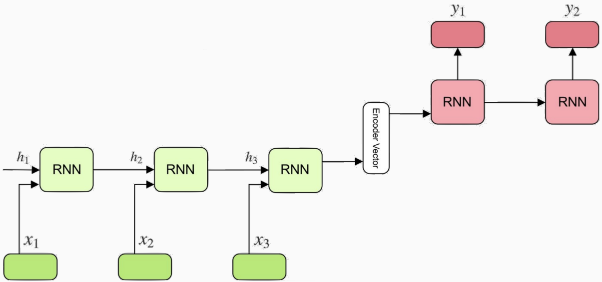
$\rightsquigarrow$ **Bidirectional recurrent neural networks**



Schuster & Paliwal (1997), Graves (2012)

Recurrent Neural Networks

The model consists of **3 parts**: Encoder, Context (encoder vector) and Decoder.

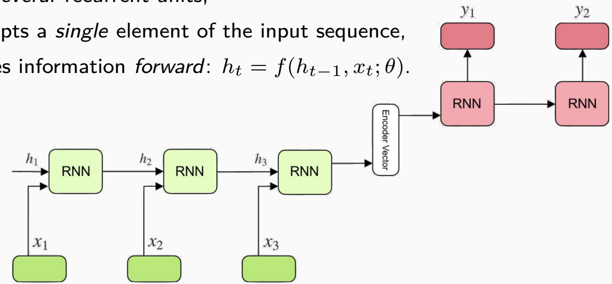


Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares,Schwenk & Bengio (2014), Sutskever, Vinyals & Le (2014)

The model consists of **3 parts**: Encoder, Context (encoder vector) and Decoder.

Encoder:

- Stack of *several* recurrent units,
- Each accepts a *single* element of the input sequence,
- Propagates information *forward*: $h_t = f(h_{t-1}, x_t; \theta)$.

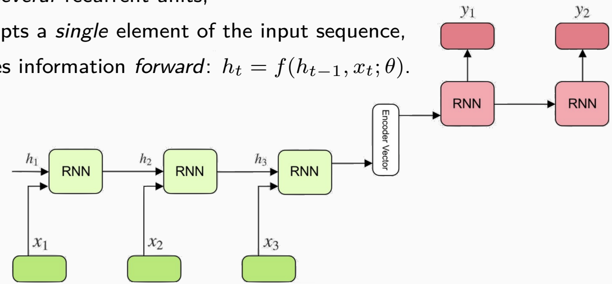


Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares,Schwenk & Bengio (2014), Sutskever, Vinyals & Le (2014)

The model consists of **3 parts**: Encoder, Context (encoder vector) and Decoder.

Encoder:

- Stack of *several* recurrent units,
- Each accepts a *single* element of the input sequence,
- Propagates information *forward*: $h_t = f(h_{t-1}, x_t; \theta)$.



Context:

- Final hidden state produced from the encoder
- Goal: Encapsulate the information for all input elements

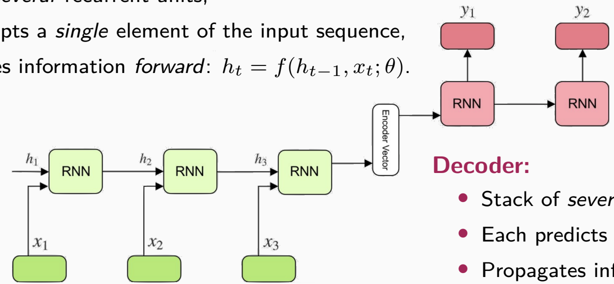
Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares,Schwenk & Bengio (2014), Sutskever, Vinyals & Le (2014)

Sequence-to-Sequence Model

The model consists of **3 parts**: Encoder, Context (encoder vector) and Decoder.

Encoder:

- Stack of *several* recurrent units,
- Each accepts a *single* element of the input sequence,
- Propagates information *forward*: $h_t = f(h_{t-1}, x_t; \theta)$.



Decoder:

- Stack of *several* recurrent units,
- Each predicts an output y_t
- Propagates information *forward*: $h_t = g(h_{t-1}, \zeta)$.

Context:

- Final hidden state produced from the encoder
- Goal: Encapsulate the information for all input elements

Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares,Schwenk & Bengio (2014), Sutskever, Vinyals & Le (2014)

RNN: Three blocks of parameter/transformations:

1. `Input` $x_t \longmapsto$ `Hidden state` h_t
2. Previous `hidden state` $h_t \longmapsto$ Next `hidden state` h_{t+1}
3. `Hidden state` $h_t \longmapsto$ `Output` y_t

RNN: Three blocks of parameter/transformations:

1. `Input` $x_t \longmapsto$ `Hidden state` h_t
2. Previous `hidden state` $h_t \longmapsto$ Next `hidden state` h_{t+1}
3. `Hidden state` $h_t \longmapsto$ `Output` y_t

RNN architecture: Each of the blocks is associated with a single weight matrix

$\longleftrightarrow$ Single layer within a deep MLP

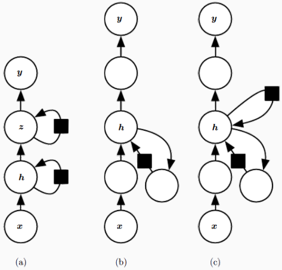
RNN: Three blocks of parameter/transformations:

1. Input $x_t \longmapsto$ Hidden state h_t
2. Previous hidden state $h_t \longmapsto$ Next hidden state h_{t+1}
3. Hidden state $h_t \longmapsto$ Output y_t

RNN architecture: Each of the blocks is associated with a single weight matrix

$\longleftrightarrow$ Single layer within a deep MLP

$\rightsquigarrow$ ***We can introduce depth into each of these operations!***



Deep Learning, Goodfellow, Bengio, Courville (2016)
§ 10.5 Deep Recurrent Networks

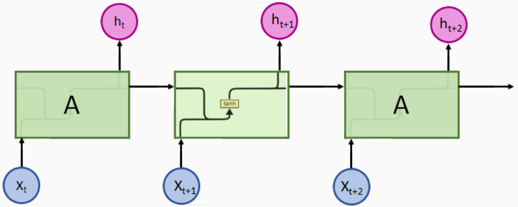
www.deeplearningbook.org

Training Recurrent Networks

Forward Propagation (Input & output of same length)

Weight matrices: Connections between the different states

Activation functions:

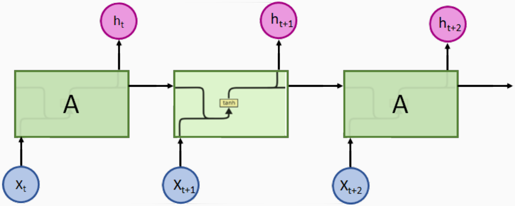


Forward Propagation (Input & output of same length)

Weight matrices: Connections between the different states

1. `Input-to-Hidden`: $\boldsymbol{U}$
2. `Hidden-to-Hidden`: $\boldsymbol{W}$ $+$ Bias vector : $\boldsymbol{b}$ ans $\boldsymbol{c}$.
3. `Hidden-to-Output`: $\boldsymbol{V}$

Activation functions:

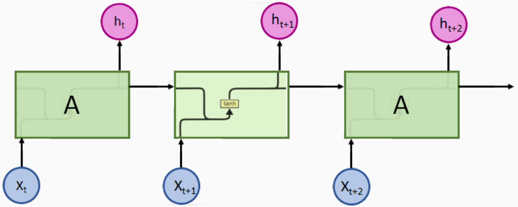


Weight matrices: Connections between the different states

1. `Input-to-Hidden`: $\boldsymbol{U}$
2. `Hidden-to-Hidden`: $\boldsymbol{W}$ $\quad + \quad$ Bias vector : $\boldsymbol{b}$ ans $\boldsymbol{c}$.
3. `Hidden-to-Output`: $\boldsymbol{V}$

Activation functions:

1. `Input-to-Hidden`: Hyperbolic tangent activation function
2. Assume that the output is discrete.
   Outputs $\longleftrightarrow$ Unnormalized probabilities of each possible value
   $\rightsquigarrow$ Softmax aftivation function to obtain normalized probabilities



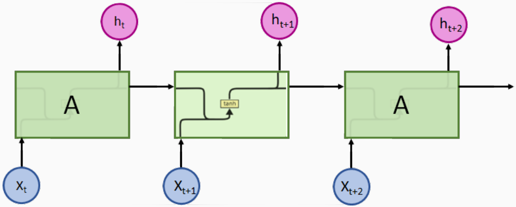
Forward Propagation (Input & output of same length)

Weight matrices: Connections between the different states

1. `Input-to-Hidden`: $\boldsymbol{U}$
2. `Hidden-to-Hidden`: $\boldsymbol{W}$ $\quad + \quad$ Bias vector : $\boldsymbol{b}$ ans $\boldsymbol{c}$.
3. `Hidden-to-Output`: $\boldsymbol{V}$

Activation functions:

1. `Input-to-Hidden`: Hyperbolic tangent activation function
2. Assume that the output is discrete.
   Outputs $\longleftrightarrow$ Unnormalized probabilities of each possible value
   $\leadsto$ Softmax aftivation function to obtain normalized probabilities



Given initial state h_0,
For each step $t \in [\![1, T]\!]$:

$$h_t = \tanh\left(W h_{t-1} + U x_t + b\right)$$
$$\hat{y}_t = \operatorname{softmax}\left(V h_t + c\right)$$

Forward Propagation (Input & output of same length)

Weight matrices: Connections between the different states

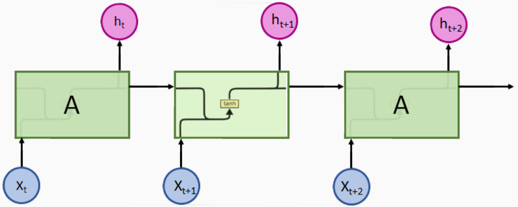
1. `Input-to-Hidden`: $\boldsymbol{U}$
2. `Hidden-to-Hidden`: $\boldsymbol{W}$ $\quad + \quad$ Bias vector : $\boldsymbol{b}$ ans $\boldsymbol{c}$.
3. `Hidden-to-Output`: $\boldsymbol{V}$

Activation functions:

1. `Input-to-Hidden`: Hyperbolic tangent activation function
2. Assume that the output is discrete.

   Outputs $\longleftrightarrow$ Unnormalized probabilities of each possible value

   $\rightsquigarrow$ Softmax aftivation function to obtain normalized probabilities



Given initial state h_0,

For each step $t \in [\![1, T]\!]$:

$$h_t = \tanh\left(W h_{t-1} + U x_t + b\right)$$
$$\hat{y}_t = \operatorname{softmax}\left(V h_t + c\right)$$

Training Recurrent Networks

Computing the Gradient in a Recurrent Neural Network

Total loss function for a given sequence values x paired with a sequence of y

$$\begin{aligned}\mathcal{L}\left(\{x_1,\ldots,x_T\},\{y_1,\ldots,y_T\};\theta\right) &= \sum_{t=1}^{T} L_t \quad ; \quad \theta=(U,V,W,b,c) \\ &= \sum_{t=1}^{T} -\log p_{\text{model}}(y_t|\{x_1,\ldots,x_t\})\end{aligned}$$

Computing the Gradient in a Recurrent Neural Network

Total loss function for a given sequence values x paired with a sequence of y

$$\mathcal{L}\left(\{x_1, \ldots, x_T\}, \{y_1, \ldots, y_T\}; \theta\right) = \sum_{t=1}^{T} L_t \quad ; \quad \theta = (U, V, W, b, c)$$

$$= \sum_{t=1}^{T} -\log p_{\text{model}}(y_t | \{x_1, \ldots, x_t\})$$

Gradient of this loss function expensive operation:

1. A **forward** propagation pass moving left to right,
2. Followed by a **backward** propagation pass moving right to left.

$\Longrightarrow$ $\boxed{\text{Runtime} = O(T)}$ and cannot be reduced by parallelization...

Computing the Gradient in a Recurrent Neural Network

Total loss function for a given sequence values x paired with a sequence of y

$$\begin{aligned}\mathcal{L}\left(\{x_1,\ldots,x_T\},\{y_1,\ldots,y_T\};\theta\right) &= \sum_{t=1}^{T} L_t \quad ; \quad \theta=(U,V,W,b,c)\\ &= \sum_{t=1}^{T} -\log p_{\text{model}}(y_t|\{x_1,\ldots,x_t\})\end{aligned}$$

Gradient of this loss function expensive operation:

1. A **forward** propagation pass moving left to right,
2. Followed by a **backward** propagation pass moving right to left.

$\Longrightarrow$ $\boxed{\text{Runtime} = O(T)}$ and cannot be reduced by parallelization...

Moreover, states computed in the forward pass must be stored until they are reused during the backward pass

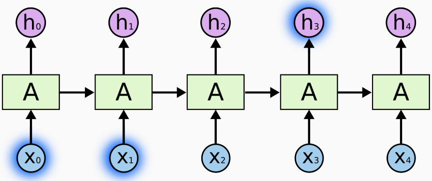
$\Longrightarrow$ $\boxed{\text{Memory cos} = O(T)}$

For an example: *Deep Learning*, Goodfellow et al. (2016), § 10.2.2

Training Recurrent Networks

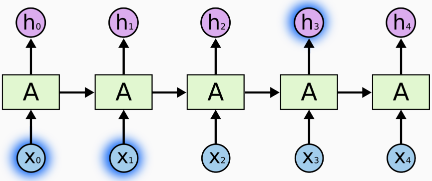
Strenght of RNNs: Being able to connect previous information to the present task

If the gap between relevant information and where it is needed is small, RNNs work "perfectly"

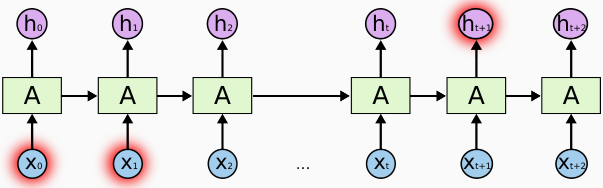


Strenght of RNNs: Being able to connect previous information to the present task

☺ If the gap between relevant information and where it is needed is small, RNNs work "perfectly"



☹ Unfortunately, as this gap widens, RNNs become unable to learn to connect information...



Vanishing of the Gradients

Basic problem: Gradients propagated over many stages

- **vanish** (most of the time)
- or **explode** (rarely, but with much damage to the optimization)

And even without it, *long-term dependencies arises from exponentially smaller weights given to long-term interactions compared to short-term ones*.

Vanishing of the Gradients

Basic problem: Gradients propagated over many stages

- **vanish** (most of the time)
- or **explode** (rarely, but with much damage to the optimization)

And even without it, *long-term dependencies arises from exponentially smaller weights given to long-term interactions compared to short-term ones*.

RNN $\Longrightarrow$ Composition of the same function multiple times, once per time step.

$\rightsquigarrow$ **Extremely nonlinear behavior**

Vanishing of the Gradients

Basic problem: Gradients propagated over many stages

- **vanish** (most of the time)
- or **explode** (rarely, but with much damage to the optimization)

And even without it, *long-term dependencies arises from exponentially smaller weights given to long-term interactions compared to short-term ones*.

RNN $\implies$ Composition of the same function multiple times, once per time step.
$\rightsquigarrow$ **Extremely nonlinear behavior**

Toy model: No input, no activation function: $h_t = W^\top h_{t-1}$

Assume that W admits an eigendecomposition.

Vanishing of the Gradients

Basic problem: Gradients propagated over many stages

- **vanish** (most of the time)
- or **explode** (rarely, but with much damage to the optimization)

And even without it, *long-term dependencies arises from exponentially smaller weights given to long-term interactions compared to short-term ones.*

RNN $\Longrightarrow$ Composition of the same function multiple times, once per time step.

$\leadsto$ **Extremely nonlinear behavior**

Toy model: No input, no activation function: $h_t = W^\top h_{t-1}$

Assume that W admits an eigendecomposition. Then $\boxed{h_t = P^\top \Lambda^t P h_0}$

Vanishing of the Gradients

Basic problem: Gradients propagated over many stages

- **vanish** (most of the time)
- or **explode** (rarely, but with much damage to the optimization)

And even without it, *long-term dependencies arises from exponentially smaller weights given to long-term interactions compared to short-term ones*.

RNN $\implies$ Composition of the same function multiple times, once per time step.
$\leadsto$ **Extremely nonlinear behavior**

Toy model: No input, no activation function: $h_t = W^\top h_{t-1}$

Assume that W admits an eigendecomposition. Then $\boxed{h_t = P^\top \Lambda^t P h_0}$

$|\lambda| < 1$ decay to zero

$|\lambda| > 1$ explosion

$\implies$ *Any component of h_0 not aligned with the largest eigenvector will eventually be discarded*

Vanishing of the Gradients

Basic problem: Gradients propagated over many stages

- **vanish** (most of the time)
- or **explode** (rarely, but with much damage to the optimization)

And even without it, *long-term dependencies arises from exponentially smaller weights given to long-term interactions compared to short-term ones*.

RNN $\Longrightarrow$ Composition of the same function multiple times, once per time step.
$\rightsquigarrow$ **Extremely nonlinear behavior**

Toy model: No input, no activation function: $h_t = W^\top h_{t-1}$

Assume that W admits an eigendecomposition. Then $\boxed{h_t = P^\top \Lambda^t P h_0}$

$|\lambda| < 1$ decay to zero
$|\lambda| > 1$ explosion
$\Longrightarrow$ *Any component of h_0 not aligned with the largest eigenvector will eventually be discarded*

Remark: **Problem particular to recurrent networks**, due to the multiple composition of same function

Hochreiter(1991), Bengio, Simard & Frasconi (1994)

Next Course: Some approaches to reduce this difficulty

BUT: *Problem of learning long-term dependencies remains one of the main challenges in deep learning*

Appendix: Categorical Variable Encoding

Encoding: Convert a categorical variable to numerical values for machine learning model building.

Different types of Encoding:

Micci-Barreca (2001)

- Label Encoding
- One Hot Encoding
- Target Encoding

Feature	Feature
A	A
B	B
C	C
B	B
C	C
A	A
A	A
C	C
B	B
B	B
C	C
A	A

Target	Feature
0.39	A
0.24	B
2.21	C
0.31	B
0.76	C
-0.74	A
0.27	A
4.01	C
2.28	B
0.19	B
2.03	C
-0.05	A

Credit: Brendan Hasz blog post

An others: Word2Vec, BERT (Bidirectional Encoder Representations from Transformers), . .

Encoding: Convert a categorical variable to numerical values for machine learning model building.

Different types of Encoding:

Micci-Barreca (2001)

- Label Encoding
- One Hot Encoding
- Target Encoding

Feature	Feature
0	A
1	B
2	C
1	B
2	C
0	A
0	A
2	C
1	B
1	B
2	C
0	A

Target	Feature
0.39	A
0.24	B
2.21	C
0.31	B
0.76	C
-0.74	A
0.27	A
4.01	C
2.28	B
0.19	B
2.03	C
-0.05	A

Credit: Brendan Hasz blog post

An others: Word2Vec, BERT (Bidirectional Encoder Representations from Transformers), . .

Appendix: Categorical Variable Encoding

Encoding: Convert a categorical variable to numerical values for machine learning model building.

Different types of Encoding:

Micci-Barreca (2001)

- Label Encoding
- One Hot Encoding
- Target Encoding

Feature
0
1
2
1
2
0
0
2
1
1
2
0

Feature_A	Feature_B	Feature_C
1	0	0
0	1	0
0	0	1
0	1	0
0	0	1
1	0	0
1	0	0
0	0	1
0	1	0
0	1	0
0	0	1
1	0	0

Target
0.39
0.24
2.21
0.31
0.76
-0.74
0.27
4.01
2.28
0.19
2.03
-0.05

Feature
A
B
C
B
C
A
A
C
B
B
C
A

Credit: Brendan Hasz blog post

An others: Word2Vec, BERT (Bidirectional Encoder Representations from Transformers), . .

Appendix: Categorical Variable Encoding

Encoding: Convert a categorical variable to numerical values for machine learning model building.

Different types of Encoding:

Micci-Barreca (2001)

- Label Encoding
- One Hot Encoding
- Target Encoding

Feature
0
1
2
1
2
0
0
2
1
1
2
0

Feature_A	Feature_B	Feature_C
1	0	0
0	1	0
0	0	1
0	1	0
0	0	1
1	0	0
1	0	0
0	0	1
0	1	0
0	1	0
0	0	1
1	0	0

Target	Feature
0.39	-0.03
0.24	0.76
2.21	2.25
0.31	0.76
0.76	2.25
-0.74	-0.03
0.27	-0.03
4.01	2.25
2.28	0.76
0.19	0.76
2.03	2.25
-0.05	-0.03

Credit: Brendan Hasz blog post

An others: Word2Vec, BERT (Bidirectional Encoder Representations from Transformers), . .

Bengio, Y., Simard, P., and Frasconi, P. (1994). **Learning long-term dependencies with gradient descent is difficult.** IEEE transactions on neural networks, 5(2):157–166.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). **Learning phrase representations using rnn encoder-decoder for statistical machine translation.** arXiv preprint arXiv:1406.1078.

Elman, J. L. (1990). **Finding structure in time.** Cognitive science, 14(2):179–211.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). **Deep learning.** MIT press.

Graves, A. (2012). **Supervised sequence labelling.** In Supervised sequence labelling with recurrent neural networks, pages 5–13. Springer.

Hochreiter, S. (1991). **Untersuchungen zu dynamischen neuronalen netzen.** Diploma, Technische Universität München, 91(1).

Hopfield, J. J. (1982). **Neural networks and physical systems with emergent collective computational abilities.** Proceedings of the national academy of sciences, 79(8):2554–2558.

Little, W. A. (1974). **The existence of persistent states in the brain.** Mathematical biosciences, 19(1-2):101–120.

Maier, A. (2020). **Lecture notes on Deep Learning.** Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU).

References ii

Micci-Barreca, D. (2001). **A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems.** ACM SIGKDD Explorations Newsletter, 3(1):27–32.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). **Learning representations by back-propagating errors.** nature, 323(6088):533–536.

Schuster, M. and Paliwal, K. K. (1997). **Bidirectional recurrent neural networks.** IEEE transactions on Signal Processing, 45(11):2673–2681.

Siegelmann, H. T. and Sontag, E. D. (1995). **On the computational power of neural nets.** Journal of computer and system sciences, 50(1):132–150.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). **Sequence to sequence learning with neural networks.** In Advances in neural information processing systems, pages 3104–3112.