Sequence Modeling: Recurrent Neural Networks

High Dimensional and Deep Learning – juliette.chevallier@insa-toulouse.fr
N7 & INSA Toulouse, 5th year ModIA

Sequence Modeling

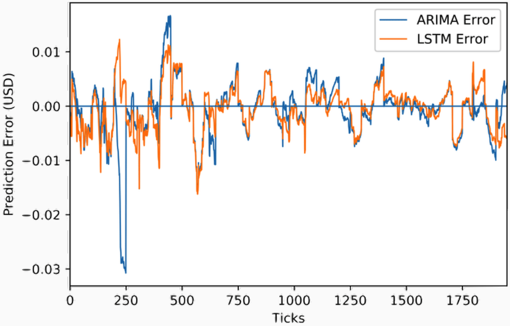
Sequential Data

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.

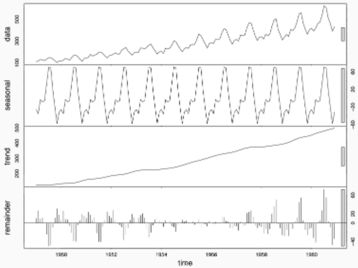
From: Vinyals, Toshev, Bengio, Erhan. Show and tell: A neural image caption generator. *CVPR* 2015

Do not get rid of traditional models too quickly!



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

Sequence Modeling



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

ARIMA *vs.* (Recurrent) Neural Network

ARIMA:	**Neural Network:**
+ 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	

ARIMA *vs.* (Recurrent) Neural Network

ARIMA:	**Neural Network:**
+ Out-perform deep learning methods for forecasting **short-term** + Out-perform deep learning methods for forecasting on **univariate** data	+ Ability to learn noisy and non-linear relationships, regardless of the number of inputs + Relax univariate assumption
– Focus on univariate data, with linear relationships, and fixed and manually-diagnosed temporal dependence	

ARIMA *vs.* (Recurrent) Neural Network

ARIMA:	Neural Network:
+ Out-perform deep learning methods for forecasting **short-term** + Out-perform deep learning methods for forecasting on **univariate** data	+ Ability to learn noisy and non-linear relationships, regardless of the number of inputs + Relax univariate assumption
– Focus on univariate data, with linear relationships, and fixed and manually-diagnosed temporal dependence	– More complicated and difficult to train – Do not exceed the performance of ARIMA in most cases

ARIMA *vs.* (Recurrent) Neural Network

ARIMA:	**Neural Network:**
+ Out-perform deep learning methods for forecasting **short-term** + Out-perform deep learning methods for forecasting on **univariate** data	+ Ability to learn noisy and non-linear relationships, regardless of the number of inputs + Relax univariate assumption
– Focus on univariate data, with linear relationships, and fixed and manually-diagnosed temporal dependence	– More complicated and difficult to train – Do not exceed the performance of ARIMA in most cases

$\rightsquigarrow$ **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?*

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

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)

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 !)

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.

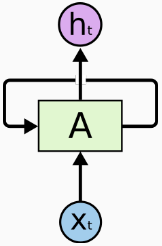
- Traditional neural networks can't do this.

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

- 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

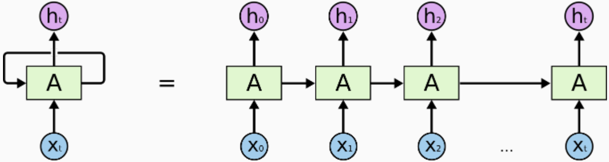
 # Output h_t at time t

 # 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

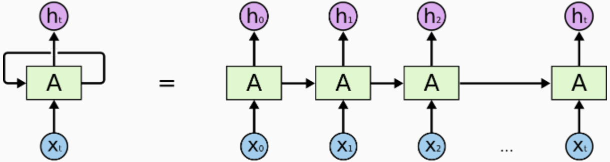
- RNN $\equiv$ Multiple copies of the **same** network,
  each passing a message to its successor: $h_t = \boldsymbol{f}(h_{t-1}, x_t; \theta)$.



- **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

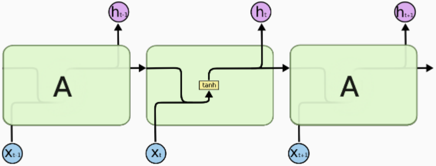
- RNN $\equiv$ Multiple copies of the **same** network,
  each passing a message to its successor: $h_t = \boldsymbol{f}(h_{t-1}, x_t; \theta)$.



- **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*

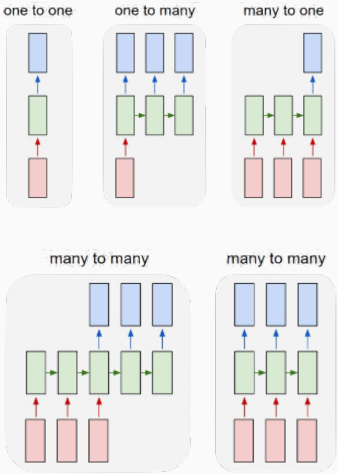
- *on all time steps,*
- *and all sequence lengths.*



Standard Recurrent Neural Networks

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

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

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.*

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

Bidirectional RNNs

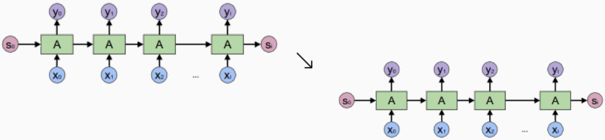
- All the recurrent networks we have considered so far have a causal structure:

  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.*

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



Schuster & Paliwal (1997), Graves (2012)

Bidirectional RNNs

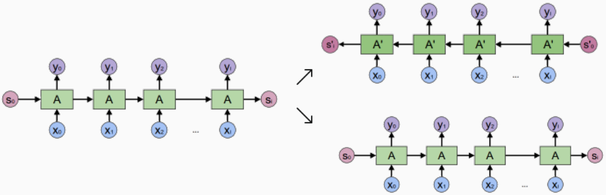
- All the recurrent networks we have considered so far have a causal structure:

  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)

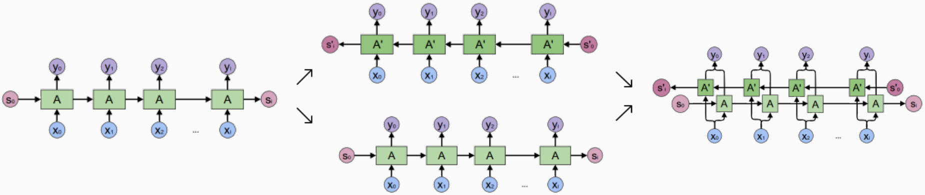
- All the recurrent networks we have considered so far have a causal structure:

  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

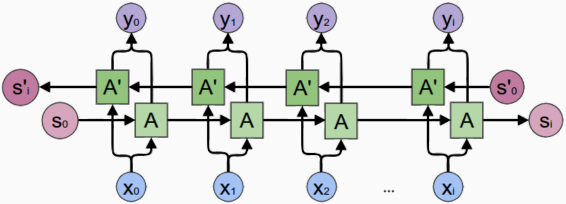
- All the recurrent networks we have considered so far have a causal structure:

  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**

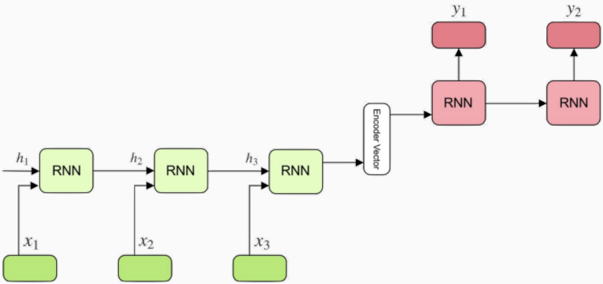


Schuster & Paliwal (1997), Graves (2012)

Recurrent Neural Networks

Sequence-to-Sequence Model

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



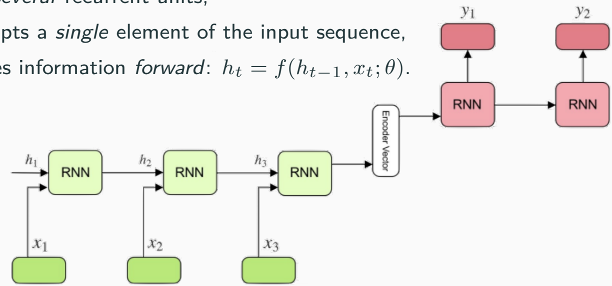
Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares,Schwenk & Bengio (2014), Sutskever, Vinyals & Le (2014)

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*: $h_t = f(h_{t-1}, x_t; \theta)$.



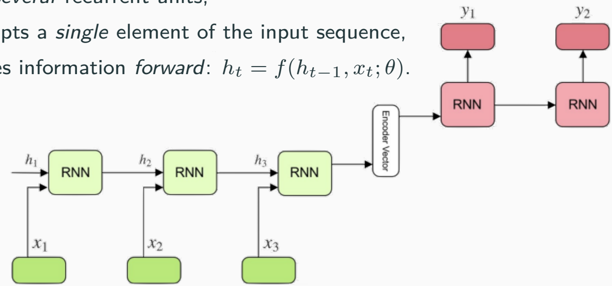
Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares,Schwenk & Bengio (2014), Sutskever, Vinyals & Le (2014)

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*: $h_t = f(h_{t-1}, x_t; \theta)$.



Context:

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

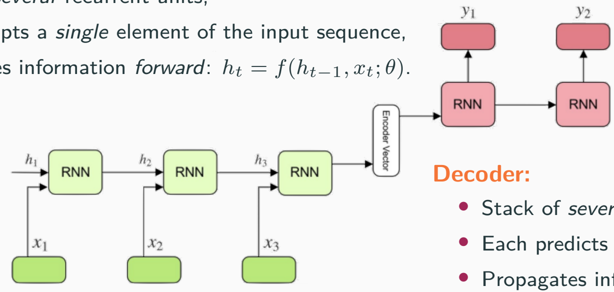
Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares,Schwenk & Bengio (2014), Sutskever, Vinyals & Le (2014)

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)

Deep Recurrent Networks

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

Deep Recurrent Networks

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

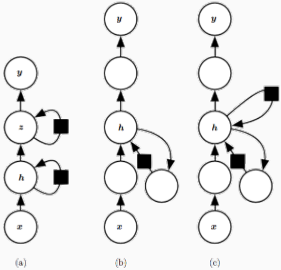
Deep Recurrent Networks

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

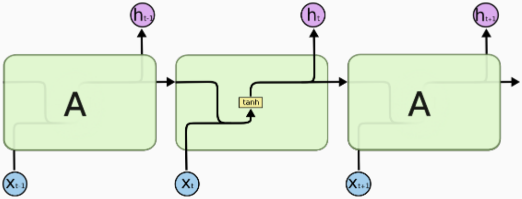
www.deeplearningbook.org

Training Recurrent Networks

Forward Propagation (Input & output of same length)

Weight matrices: Connections between the different states

Activation functions:

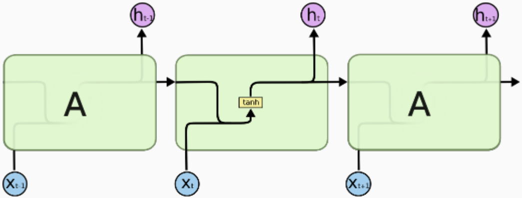


Forward Propagation (Input & output of same length)

Weight matrices: Connections between the different states

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

Activation functions:



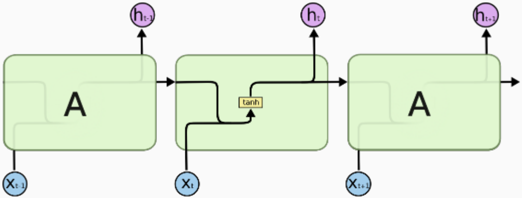
Forward Propagation (Input & output of same length)

Weight matrices: Connections between the different states

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



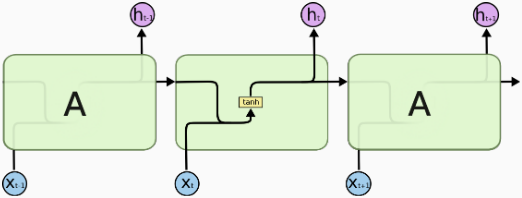
Forward Propagation (Input & output of same length)

Weight matrices: Connections between the different states

1. `Input-to-Hidden`: $\boldsymbol{U}$
2. `Hidden-to-Hidden`: $\boldsymbol{W}$ $\quad + \quad$ Bias vector : $\boldsymbol{b}$ and $\boldsymbol{c}$.
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 $t \in [\![1, T]\!]$:

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

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

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

Computing the Gradient in a Recurrent Neural Network

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

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

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...

Computing the Gradient in a Recurrent Neural Network

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

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

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

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

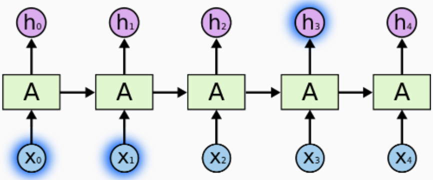
For an example: *Deep Learning*, Goodfellow et al. (2016), § 10.2.2

Training Recurrent Networks

Long-Term Dependencies

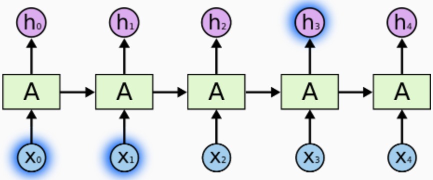
Strenght of RNNs: Being able to connect previous information to the present task

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

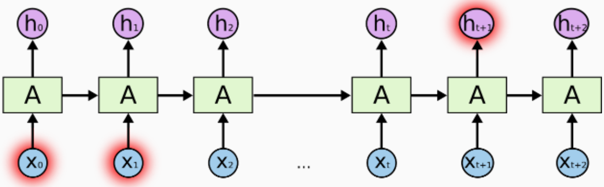


Strenght of RNNs: Being able to connect previous information to the present task

☺ 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...

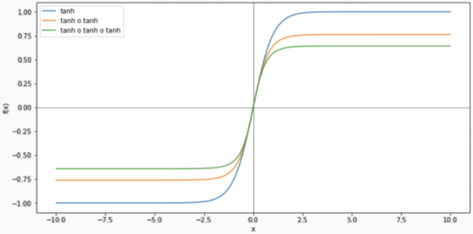


A Short Memory Model

- The hyperbolic tangent tends to *smash* the values taken as input.
- Information coming from x_1 passes t times through $\tanh$

  $\rightsquigarrow$ x_1 *is very crushed.*
- At the same time, the one from x_t passes through $\tanh$ only once

  $\rightsquigarrow$ x_t *is preserved overall.*



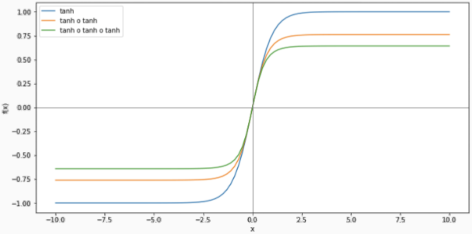
A Short Memory Model

- The hyperbolic tangent tends to *smash* the values taken as input.

- Information coming from x_1 passes t times through $\tanh$
  $\rightsquigarrow$ *x_1 is very crushed.*

- At the same time, the one from x_t passes through $\tanh$ only once
  $\rightsquigarrow$ *x_t is preserved overall.*

$\Longrightarrow$ $x_t \gg x_1$ for predictions

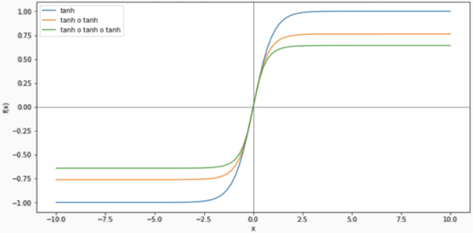


A Short Memory Model

- The hyperbolic tangent tends to *smash* the values taken as input.

- Information coming from x_1 passes t times through $\tanh$
  $\rightsquigarrow$ x_1 *is very crushed.*
- At the same time, the one from x_t passes through $\tanh$ only once
  $\rightsquigarrow$ x_t *is preserved overall.*

$\Longrightarrow$ $x_t \gg x_1$ for predictions



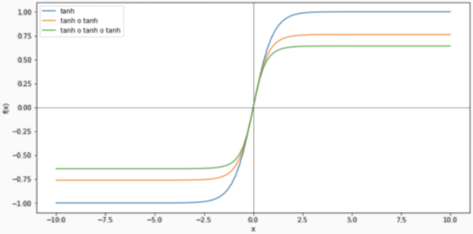
Naive approach: Compensate this crushing effect by a stronger weight associated with h_1, via the weight matrix W.

A Short Memory Model

- The hyperbolic tangent tends to *smash* the values taken as input.

- Information coming from x_1 passes t times through $\tanh$
  $\rightsquigarrow$ *x_1 is very crushed.*
- At the same time, the one from x_t passes through $\tanh$ only once
  $\rightsquigarrow$ *x_t is preserved overall.*

$\Longrightarrow$ $x_t \gg x_1$ for predictions



Naive approach: Compensate this crushing effect by a stronger weight associated with h_1, via the weight matrix W.

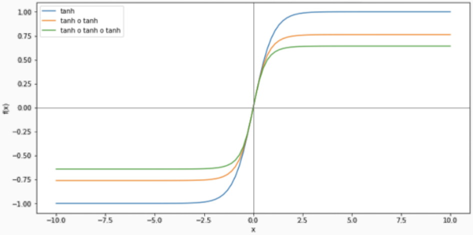
But: Matrix W shared by all cells in the layer, *i.e.* all intermediate outputs weighted by the same weight.

- The hyperbolic tangent tends to *smash* the values taken as input.

- Information coming from x_1 passes t times through $\tanh$
  $\rightsquigarrow$ *x_1 is very crushed.*

- At the same time, the one from x_t passes through $\tanh$ only once
  $\rightsquigarrow$ *x_t is preserved overall.*

$\Longrightarrow$ $x_t \gg x_1$ for predictions



Naive approach: Compensate this crushing effect by a stronger weight associated with h_1, via the weight matrix W.

But: Matrix W shared by all cells in the layer, *i.e.* all intermediate outputs weighted by the same weight.

$\rightsquigarrow$ **LSTM, GRU, ...**

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 $\implies$ 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 eigen decomposition.

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 eigen decomposition. 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.
$\rightsquigarrow$ **Extremely nonlinear behavior**

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

Assume that W admits an eigen decomposition. 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 $\implies$ Composition of the same function multiple times, once per time step.
$\rightsquigarrow$ **Extremely nonlinear behavior**

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

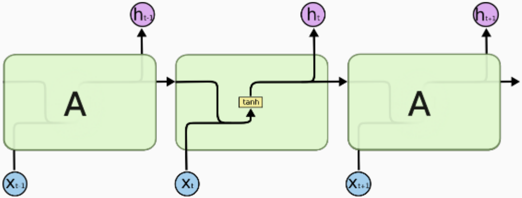
Assume that W admits an eigen decomposition. Then $\boxed{h_t = P^\top \Lambda^t P h_0}$

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

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

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

Summary of the RNN Framework



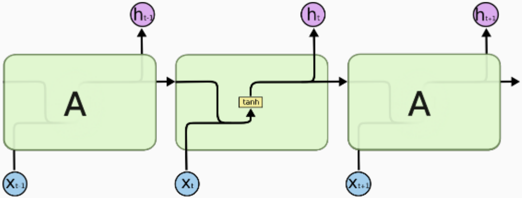
Given initial state h_0,
For each step $t \in [\![1, T]\!]$:

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

Summary of the RNN Framework

Advantages of RNN:

- Performance not significantly affected from *missing values*;
- Can find complex patterns in the input time series;
- Good results in forecasting more then few-steps;
- Can model sequence of data so that each sample can be assumed to be dependent on previous ones.



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

$$
\boxed{\begin{aligned} h_t &= \tanh\left(W h_{t-1} + U x_t + b\right) \\ \hat{y}_t &= \mathrm{softmax}\left(V h_t + c\right) \end{aligned}}
$$

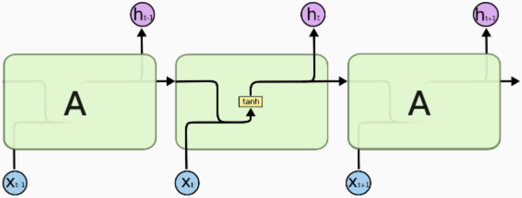
Summary of the RNN Framework

Advantages of RNN:

- Performance not significantly affected from *missing values*;
- Can find complex patterns in the input time series;
- Good results in forecasting more then few-steps;
- Can model sequence of data so that each sample can be assumed to be dependent on previous ones.

Disadvantages of RNN:

- When trained on long time series, RNNs suffer from the vanishing gradient or exploding gradient problem;
- Weak memory unable to take into account several elements of the past in the prediction of the future;
- Training of a Recurrent Neural Network hard to parallelize and computationally expensive.



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

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

Challenge of Long-Term Dependencies

Optimization for Long-Term Dependencies

Clipping gradient: avoids gradient explode but **NOT** gradient vanish.

- Mikolov (2012): Clip the parameter gradient from a mini-batch element-wise,
- Pascanu et al. (2013): Clip the norm of the gradient just before the parameter update.

Regularizing to encourage the information flow: Favor gradient vector being back-propagated to maintain its magnitude. (Pascanu et al., 2013)

Optimization for Long-Term Dependencies

Clipping gradient: avoids gradient explode but **NOT** gradient vanish.

- Mikolov (2012): Clip the parameter gradient from a mini-batch element-wise,
- Pascanu et al. (2013): Clip the norm of the gradient just before the parameter update.

Regularizing to encourage the information flow: Favor gradient vector being back-propagated to maintain its magnitude. (Pascanu et al., 2013)

Remarks: These techniques are *not* quite useful nowadays: Most of the time using LSTM will solve the long-term dependency problem.

Take-Home Message:

It is often much easier to design a model that is easy to optimize than it is to design a more powerful optimization algorithm.

Challenge of Long-Term Dependencies

The Challenge of Long-Term Dependencies

First idea: Design a model that operates at multiple time scales

(i) Some parts operate at *fine-grained* time scales
  $\rightsquigarrow$ Handle small details,

(ii) Other parts operate at *coarse* time scales
  $\rightsquigarrow$ Transfer information from distant past to present

The Challenge of Long-Term Dependencies

First idea: Design a model that operates at multiple time scales

(i) Some parts operate at *fine-grained* time scales
$\rightsquigarrow$ Handle small details,

(ii) Other parts operate at *coarse* time scales
$\rightsquigarrow$ Transfer information from distant past to present

Various strategies: Skipped connections, Leaky units, Deleted connections, *etc.*

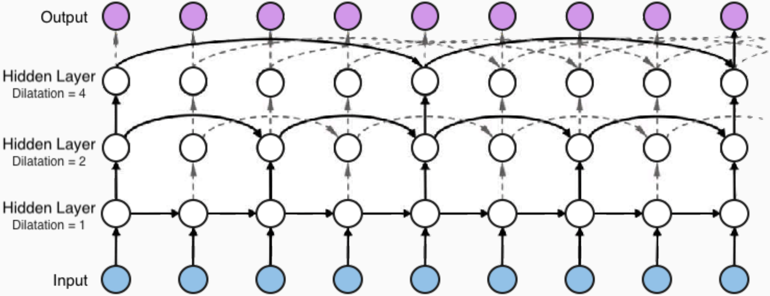
The Challenge of Long-Term Dependencies

First idea: Design a model that operates at multiple time scales

(i) Some parts operate at *fine-grained* time scales
  $\rightsquigarrow$ Handle small details,

(ii) Other parts operate at *coarse* time scales
  $\rightsquigarrow$ Transfer information from distant past to present

Various strategies: *Skipped connections*, Leaky units, Deleted connections, *etc.*



$\rightsquigarrow$ Gradients decrease exponentially with rate $\frac{T}{d}$ rather than T.

Lin, Horne, Tino and Giles (1996).

Strategies for MultipleTime Scales

Leaky Units: Mozer (1992); El Hihi & Bengio (1996).

Removing Connections: Mozer (1992); Pascanu, Mikolov & Bengio (2013).

Strategies for MultipleTime Scales

Leaky Units: Mozer (1992); El Hihi & Bengio (1996).

- **Idea**: Each hidden state h_t is a “summary of history”

Removing Connections: Mozer (1992); Pascanu, Mikolov & Bengio (2013).

Strategies for MultipleTime Scales

Leaky Units: Mozer (1992); El Hihi & Bengio (1996).

- **Idea**: Each hidden state h_t is a "summary of history", set to memorize both
  (i) a summary of the immediate *past* h_{t-1},
  (ii) and some "new stuff" of *present* time x_t.

$$\boxed{h_t = \alpha\, h_{t-1} + (1-\alpha)\, x_t}$$

Removing Connections: Mozer (1992); Pascanu, Mikolov & Bengio (2013).

Strategies for MultipleTime Scales

Leaky Units: Mozer (1992); El Hihi & Bengio (1996).

- **Idea**: Each hidden state h_t is a "summary of history", set to memorize both
  (i) a summary of the immediate *past* h_{t-1},
  (ii) and some "new stuff" of *present* time x_t.

$$\boxed{h_t = \alpha\, h_{t-1} + (1 - \alpha)\, x_t}$$

- Parameter α substitutes to the matrix W.

  $\boldsymbol{\alpha \simeq 1}$ **remembering** information about the past for a long time;

  $\boldsymbol{\alpha \simeq 0}$ information about the past is rapidly **discarded**.

Removing Connections: Mozer (1992); Pascanu, Mikolov & Bengio (2013).

Strategies for MultipleTime Scales

Leaky Units: Mozer (1992); El Hihi & Bengio (1996).

- **Idea**: Each hidden state h_t is a "summary of history", set to memorize both
  (i) a summary of the immediate *past* h_{t-1},
  (ii) and some "new stuff" of *present* time x_t.

$$\boxed{h_t = \alpha\, h_{t-1} + (1-\alpha)\, x_t}$$

- Parameter α substitutes to the matrix W.

  $\boldsymbol{\alpha \simeq 1}$ **remembering** information about the past for a long time;
  $\boldsymbol{\alpha \simeq 0}$ information about the past is rapidly **discarded**.

Removing Connections: Mozer (1992); Pascanu, Mikolov & Bengio (2013).

- **Idea**: Actively removing length-one connections and replacing them with longer connections.

Strategies for MultipleTime Scales

Leaky Units: Mozer (1992); El Hihi & Bengio (1996).

- **Idea**: Each hidden state h_t is a "summary of history", set to memorize both
  (i) a summary of the immediate *past* h_{t-1},
  (ii) and some "new stuff" of *present* time x_t.

$$\boxed{h_t = \alpha\, h_{t-1} + (1-\alpha)\, x_t}$$

- Parameter α substitutes to the matrix W.

  $\boldsymbol{\alpha \simeq 1}$ **remembering** information about the past for a long time;
  $\boldsymbol{\alpha \simeq 0}$ information about the past is rapidly **discarded**.

Removing Connections: Mozer (1992); Pascanu, Mikolov & Bengio (2013).

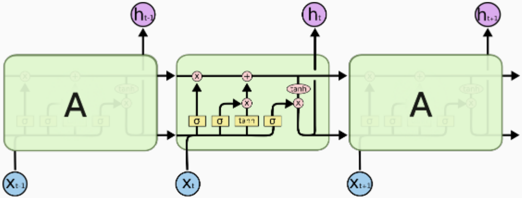
- **Idea**: Actively removing length-one connections and replacing them with longer connections.
- *Removing connections* $\neq$ *Skipping connections*:
  - *Removing connections*: Units forced to operate on along time scale;
  - *Skip connections*: Units may learn on a long time scale but may also focus on their other, short-term connections.

Gated Recurrent Neural Network

Long-Short-Term-Memory Neural Networks

"Special kind of RNN's, capable of learning long-term dependencies."

Hochreiter & Schmidhuber (1997).

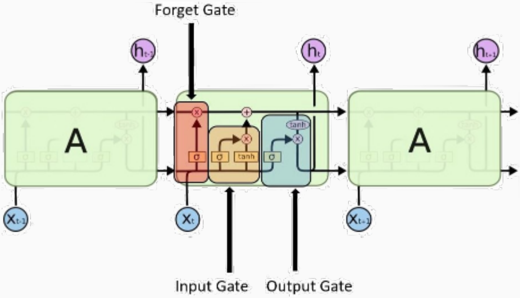


- **Idea**: divide the signal between what is important in:
  (i) *short term* through the hidden state (analogous to output of a vanilla RNN),
  (ii) *long term*, through a new state : the cell state (acts as a long term memory).

Long-Short-Term-Memory Neural Networks

"Special kind of RNN's, capable of learning long-term dependencies."

Hochreiter & Schmidhuber (1997).



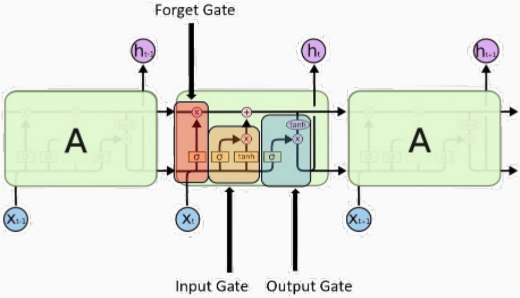
- **Idea**: divide the signal between what is important in:
  - (i) *short term* through the hidden state (analogous to output of a vanilla RNN),
  - (ii) *long term*, through a new state : the cell state (acts as a long term memory).

- LSTM has a **three** step process:

Long-Short-Term-Memory Neural Networks

"Special kind of RNN's, capable of learning long-term dependencies."

Hochreiter & Schmidhuber (1997).



- **Idea**: divide the signal between what is important in:
  (i) *short term* through the hidden state (analogous to output of a vanilla RNN),
  (ii) *long term*, through a new state : the cell state (acts as a long term memory).

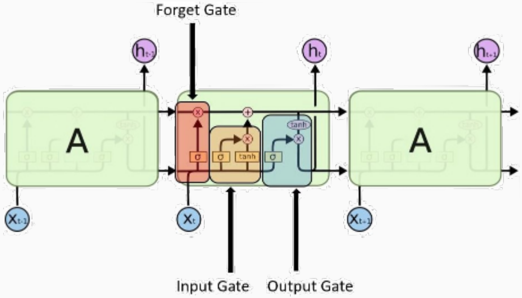
- LSTM has a **three** step process:

Forget gate From cell state, detect relevant information from the past;

Long-Short-Term-Memory Neural Networks

"Special kind of RNN's, capable of learning long-term dependencies."

Hochreiter & Schmidhuber (1997).



- **Idea**: divide the signal between what is important in:
  (i) *short term* through the hidden state (analogous to output of a vanilla RNN),
  (ii) *long term*, through a new state : the cell state (acts as a long term memory).

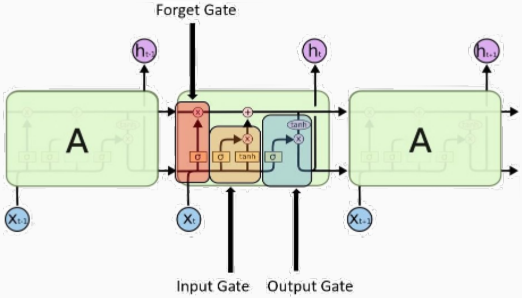
- LSTM has a **three** step process:

Forget gate From cell state, detect relevant information from the past;

Input gate From current input, select relevant in the long term information;

"Special kind of RNN's, capable of learning long-term dependencies."

Hochreiter & Schmidhuber (1997).



- **Idea**: divide the signal between what is important in:
  - (i) *short term* through the hidden state (analogous to output of a vanilla RNN),
  - (ii) *long term*, through a new state : the cell state (acts as a long term memory).

- LSTM has a **three** step process:

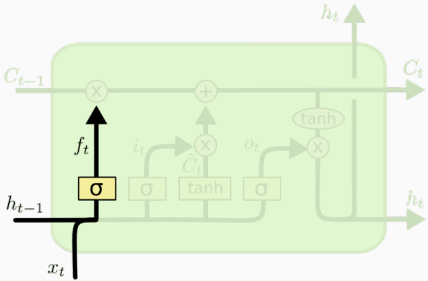
Forget gate From cell state, detect relevant information from the past;

Input gate From current input, select relevant in the long term information;

Output gate From new cell state, select important short term information to

Forget Gate

"Decides how much of the past you should remember."



$$\boxed{f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)}$$

Input:

- Previous state h_{t-1},
- Content input x_t.

Layer:

- Dense layer.

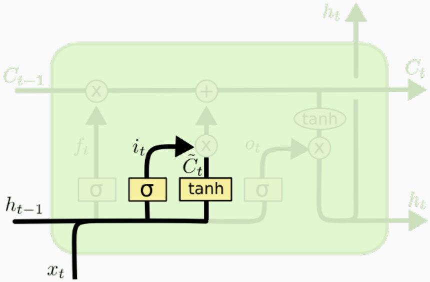
Output:

- A number between 0 (omit this) and 1 (keep this) for each number in the cell state C_{t-1}.

$\rightsquigarrow$ **Forget gate** $\equiv$ **filter** to "forget" some information of the cell state.
Term to term multiplication between f_t and C_{t-1}

$\implies$ cancellation of components of C_{t-1} whose counterparts on f_t are near 0.

"Decides how much of this unit is added to the current state."



$$\boxed{i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)}$$

$$\boxed{\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)}$$

$$\boxed{C_t = f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t}$$

Input:

- Previous state h_{t-1},
- Content input x_t.

Output:

- Cell state C_t.

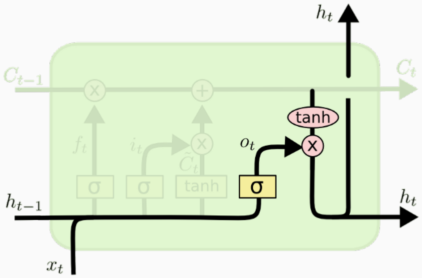
Remarks

- Input gate $i_t \equiv$ Filter,
- $\tilde{C}_t \equiv$ *Candidate* vector to update the cell state,

$\rightsquigarrow$ *Update vector* of the cell state.

Output Gate

"Decides which part of the current cell makes it to the output."



$$\boxed{o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)}$$

$$\boxed{h_t = o_t \otimes \tanh(C_t)}$$

Input:

- Previous state h_{t-1},
- Content input x_t,
- Cell state C_t,

Output:

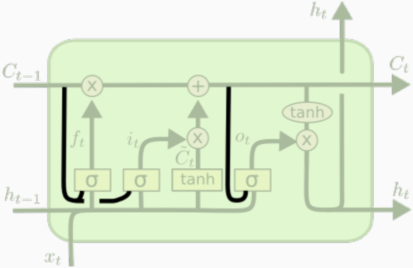
- Current state h_t.

Remarks

- Output gate $o_t \equiv$ Filter,
- New output $\equiv$ Weighted cell filtration.

Gated Recurrent Neural Network

LSTM Augmented by "Peephole Connection"



Gers & Schmidhuber (2000).

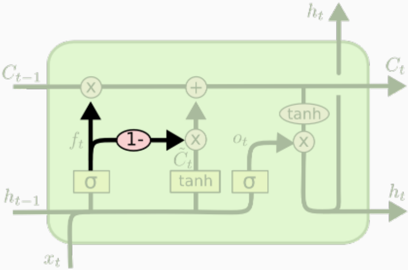
- "Peephole connections": Let the gate layers look at the cell state;
- Diagram adds peepholes to all the gates, but we can give some peepholes and not others.

$$
\begin{cases}
f_t = \sigma\left(W_f \cdot [\mathbf{C_{t-1}}, h_{t-1}, x_t] + b_f\right) \\
i_t = \sigma\left(W_i \cdot [\mathbf{C_{t-1}}, h_{t-1}, x_t] + b_i\right) \\
\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \\
C_t = f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t \\
o_t = \sigma\left(W_o \cdot [\mathbf{C_t}, h_{t-1}, x_t] + b_o\right) \\
h_t = o_t \otimes \tanh(C_t)
\end{cases}
$$

Other Minor Variants

- Coupled Input and Forget gates: $f_t = 1 - i_t$

- Full gate recurrence $f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t, C_t, f_{t-1}, i_{t-1}, o_{t-1}] + b_f\right)$



LSTM: A Search Space Odyssey (Greff et al., 2015)

- Tested the following variants, using Peephole LSTM as standard:

NIG No Input Gate,

NFG No Forget Gate,

NOG No Output Gate,

NIAF No Input Activation Function,

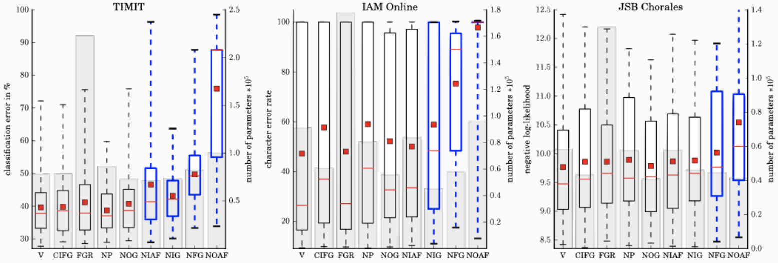
NOAF No Output Activation Function,

NP No Peepholes,

CIFG Coupled Input and Forget Gate,

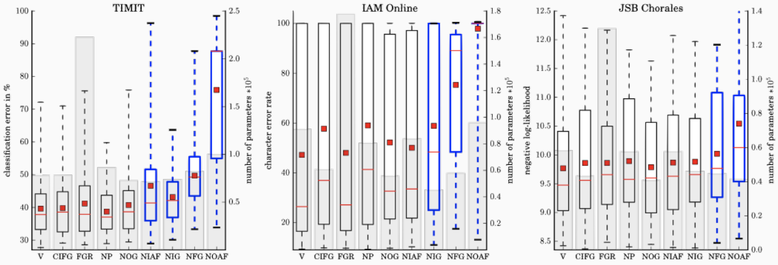
FGR Full Gate Recurrence;

- On the tasks of:
  - Timit Speech Recognition: Audio frame to 1 of 61 phonemes,
  - IAM Online Handwriting Recognition: Sketch to characters,
  - JSB Chorales: Next-step music frame prediction.



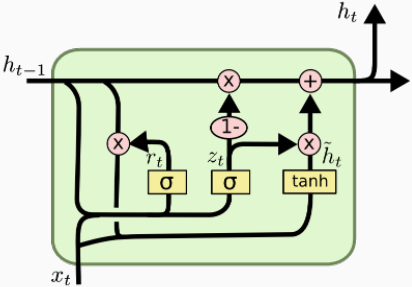
LSTM: A Search Space Odyssey (Greff et al., 2015)

- Standard LSTM performed reasonably well on multiple datasets and *none of the modifications significantly improved the performance*;
- Coupling gates and removing peephole connections simplified the LSTM without hurting performance much;
- The forget gate and output activation are crucial;
- Found interaction between learning rate and network size to be minimal – indicates calibration can be done using a small network first.



Gated Recurrent Neural Network

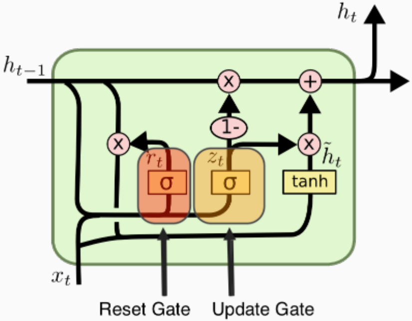
"Another special kind of RNN's, designed to solve the vanishing gradient problem."



Cho, et al. (2014).

Gated Recurrent Unit

"Another special kind of RNN's, designed to solve the vanishing gradient problem."

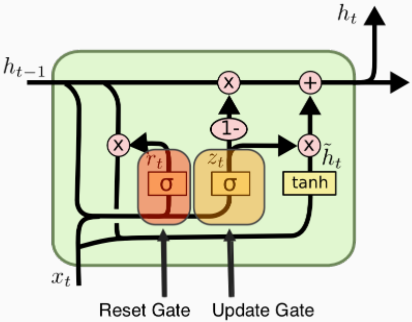


Cho, et al. (2014).

- A GRU unit is composed of:

Gated Recurrent Unit

"Another special kind of RNN's, designed to solve the vanishing gradient problem."



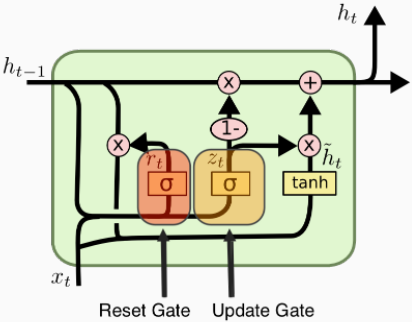
Cho, et al. (2014).

- A GRU unit is composed of:

Reset gate How much information from the previous steps can be forgotten;

Gated Recurrent Unit

"Another special kind of RNN's, designed to solve the vanishing gradient problem."



Cho, et al. (2014).

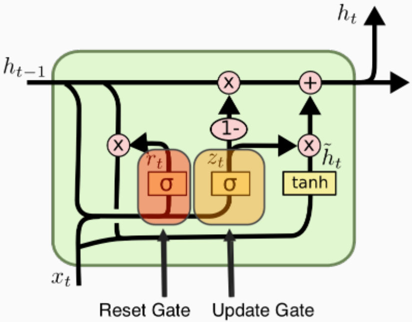
- A GRU unit is composed of:

Reset gate How much information from the previous steps can be forgotten;

Update gate How much information from the previous time steps must be saved;

Gated Recurrent Unit

"Another special kind of RNN's, designed to solve the vanishing gradient problem."



Cho, et al. (2014).

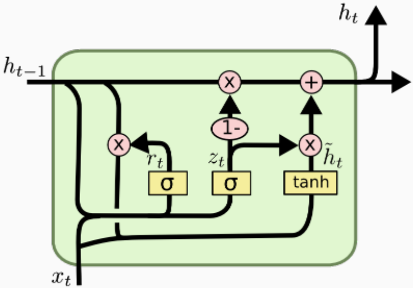
- A GRU unit is composed of:

Reset gate How much information from the previous steps can be forgotten;

Update gate How much information from the previous time steps must be saved;

Current memory Provides information throughout the sequence,
Represents the memory of the network.

Gates: Reset and Update



Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares, Schwenk & Bengio (2014).

Reset gates:

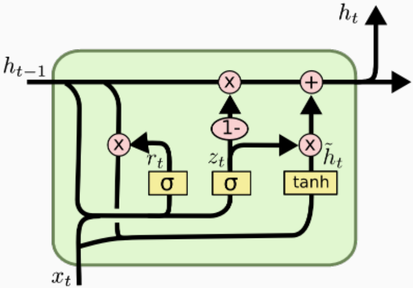
- How to combine the new input with the previous memory?
- How much information from previous time steps can be forgotten?

$$\boxed{r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t] + b_i\right)}$$

Update gates:

- How much information from previous time steps needs to be passed along to the future.

$$\boxed{z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t] + b_i\right)}$$



Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares, Schwenk & Bengio (2014).

Current memory content:

- Store the relevant information from the past

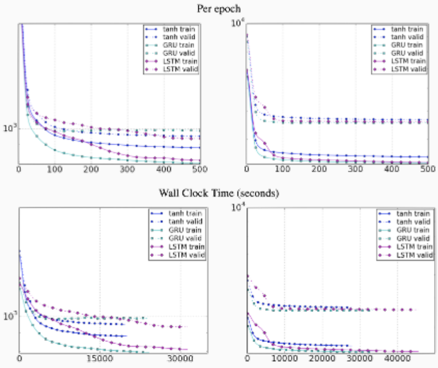
$$\boxed{\tilde{h}_t = \tanh\left(W \cdot [r_t \otimes h_{t-1}, x_t] + b_i\right)}$$

Final memory (at current time step):

- What to collect from the current memory content $\tilde{h}_t$ and what from the previous steps h_{t-1}.

$$\boxed{h_t = \sigma\left((1 - z_t \otimes h_{t-1} + z_t \otimes \tilde{h}_t\right)}$$

Vanilla RNN *vs.* LSTM *vs.* GRU



(a) Nottingham Dataset
(b) MuseData Dataset

			RNN	GRU	LSTM
Next step Music prediction	Nottingham	train	3.22	2.79	3.08
		test	**3.13**	3.23	3.20
	JSB Chorales	train	8.82	6.94	8.15
		test	9.10	**8.54**	8.67
	MuseData	train	5.64	5.06	5.18
		test	6.23	**5.99**	6.23
	Piano-midi	train	5.64	4.93	6.49
		test	9.03	**8.82**	9.03
Speech recognition	Ubisoft dataset A	train	6.29	2.31	1.44
		test	6.44	3.59	**2.70**
	Ubisoft dataset B	train	7.61	0.38	0.80
		test	7.62	**0.88**	1.26

- GRU has fewer tensor operations
  $\rightsquigarrow$ Little speedier to train than LSTM.
- No clear winner between LSTM & GRU
  $\rightsquigarrow$ Try both to determine which one works better for the considered case.

Junyoung, Caglar, KyungHyun & Bengio (2014).

A Profusion of RNN Variants

- Only a few of the most notable LSTM variants.
  Which of these variants is best? Do the differences matter?

A Profusion of RNN Variants

- Only a few of the most notable LSTM variants.
  Which of these variants is best? Do the differences matter?
  - ⇝ *Jozefowicz, et al. (2015): Empirical exploration of RNN*

A Profusion of RNN Variants

- Only a few of the most notable LSTM variants.
  Which of these variants is best? Do the differences matter?

  $\rightsquigarrow$ *Jozefowicz, et al. (2015): Empirical exploration of RNN*

- Given the rather ad-hoc design of the LSTM, the authors:
  (i) tried to determine if the architecture of LSTM is optimal,

A Profusion of RNN Variants

- Only a few of the most notable LSTM variants.
  Which of these variants is best? Do the differences matter?

  $\rightsquigarrow$ *Jozefowicz, et al. (2015): Empirical exploration of RNN*

- Given the rather ad-hoc design of the LSTM, the authors:
  (i) tried to determine if the architecture of LSTM is optimal,
  (ii) tested **10k** RNN architectures (including new ones),

A Profusion of RNN Variants

- Only a few of the most notable LSTM variants.
  Which of these variants is best? Do the differences matter?

  $\rightsquigarrow$ *Jozefowicz, et al. (2015): Empirical exploration of RNN*

- Given the rather ad-hoc design of the LSTM, the authors:
  (i) tried to determine if the architecture of LSTM is optimal,
  (ii) tested 10**k** RNN architectures (including new ones),
  (iii) found some that worked better on certain tasks.

  LSTM initialized with a large positive forget gate bias outperformed both the basic LSTM and the GRU!

A Profusion of RNN Variants

- Only a few of the most notable LSTM variants.
  Which of these variants is best? Do the differences matter?

  $\rightsquigarrow$ *Jozefowicz, et al. (2015): Empirical exploration of RNN*

- Given the rather ad-hoc design of the LSTM, the authors:
  (i) tried to determine if the architecture of LSTM is optimal,
  (ii) tested **10k** RNN architectures (including new ones),
  (iii) found some that worked better on certain tasks.

  LSTM initialized with a large positive forget gate bias outperformed both the basic LSTM and the GRU!

- All architectures are evaluated on 3 problems:

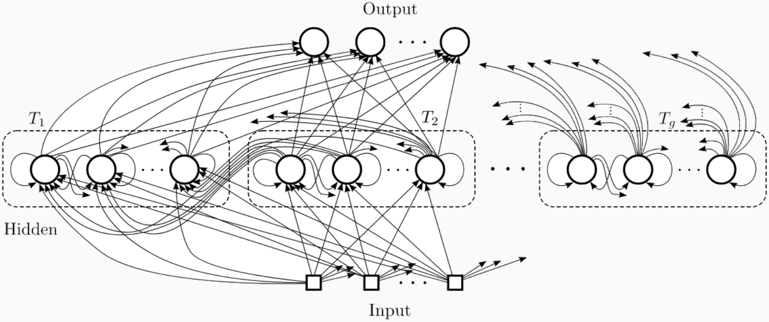
  Arithmetic Compute sum/difference of two numbers, with distractors
  3e36d9 - h1h39f94eeh43keg3c = 3369 – 13994433 = -13991064;

  XML Modeling Predict next character in valid XML modeling;

  Language Modeling Predict distributions over words (Penn.

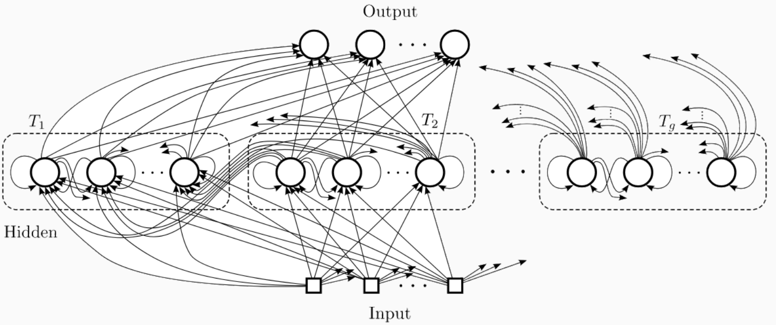
- There are also completely different strategies: *Cf.* Clockwork RNN (Koutnik, Greff, Gomez & Schmidhuber, 2014)

Clockwork Recurrent Neural Network



(Koutnik, Greff, Gomez & Schmidhuber, 2014)

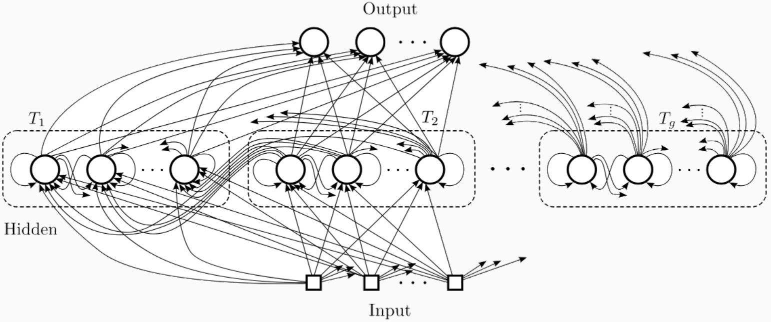
Clockwork Recurrent Neural Network



- *"Better memory" than LSTMs:* Structured hidden layer that does not enforce representing the mean of all inputs (running average in case of the LSTM).

(Koutnik, Greff, Gomez & Schmidhuber, 2014)

Clockwork Recurrent Neural Network



- *"Better memory" than LSTMs:* Structured hidden layer that does not enforce representing the mean of all inputs (running average in case of the LSTM).

- *But not really used in practice:* Need to know the numbers of modules to partition the hidden layer,
  - Depends most likely on the data,
  - No literature that suggests how to do this properly.

(Koutnik, Greff, Gomez & Schmidhuber, 2014)

Explicit Memory

Memory Networks : Pushing Further the Idea of Memory in Neural Networks

- Neural networks excel at storing *implicit knowledge*.
- However, they struggle to memorize *facts*.

Memory Networks : Pushing Further the Idea of Memory in Neural Networks

- Neural networks excel at storing *implicit knowledge*.
- However, they struggle to memorize *facts*.

  hypothesize: (Graves et al., 2014) *Neural nets lack of working memory:*
  - Stochastic gradient descent requires many presentations of the same input before it can be stored in neural network parameters,
  - And even then,that input will not be stored especially precisely.

Memory Networks : Pushing Further the Idea of Memory in Neural Networks

- Neural networks excel at storing *implicit knowledge*.
- However, they struggle to memorize *facts*.

  hypothesize: (Graves et al., 2014) *Neural nets lack of working memory:*
  - Stochastic gradient descent requires many presentations of the same input before it can be stored in neural network parameters,
  - And even then,that input will not be stored especially precisely.

$\leadsto$ *Introduction of an* ***explicit memory*** *component*.

1. **Memory networks** (Weston, Chopra & Bordes, 2014)
   - Set of memory cells that can be accessed via an addressing mechanism,
   - *But* requires a supervision signal instructing them how to use their memory cells.
2. **Neural Turing Machine** (NTM) (Graves, Wayne & Danihelka, 2014) : able to learn to *read* from and *write* arbitrary content to memory cells without explicit supervision about which actions to undertake.

Memory Networks : Pushing Further the Idea of Memory in Neural Networks

- Neural networks excel at storing *implicit knowledge*.

- However, they struggle to memorize *facts*.

  hypothesize: (Graves et al., 2014) *Neural nets lack of working memory:*
  - Stochastic gradient descent requires many presentations of the same input before it can be stored in neural network parameters,
  - And even then,that input will not be stored especially precisely.

$\rightsquigarrow$ *Introduction of an **explicit memory** component*.

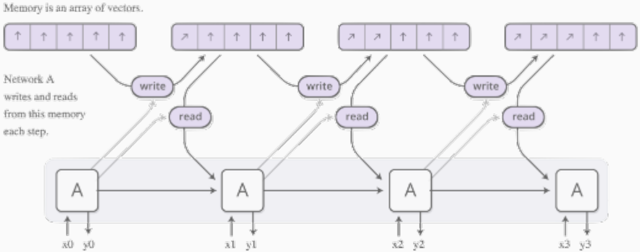
1. **Memory networks** (Weston, Chopra & Bordes, 2014)
   - Set of memory cells that can be accessed via an addressing mechanism,
   - *But* requires a supervision signal instructing them how to use their memory cells.
2. **Neural Turing Machine** (NTM) (Graves, Wayne & Danihelka, 2014) : able to learn to *read* from and *write* arbitrary content to memory cells without explicit supervision about which actions to undertake.

- *Memory cell in NTM* $\longleftrightarrow$ *Memory cells in LSTMs and GRUs.*
  NTM outputs an internal state that chooses which cell to read from or write to, (think of memory accesses in a computer read from or write to a specific address).

Explicit Memory

Neural Turing Machines

- Designed such that every component of the architecture is differentiable, making it straight forward to train with gradient descent.

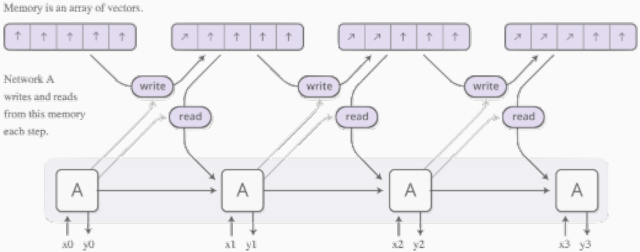


Graves et al. (2014). See also `https://distill.pub/2016/augmented-rnns/`

Neural Turing Machines

- Designed such that every component of the architecture is differentiable, making it straight forward to train with gradient descent.

- Controller: LSTM *or* Feedforward neural net.

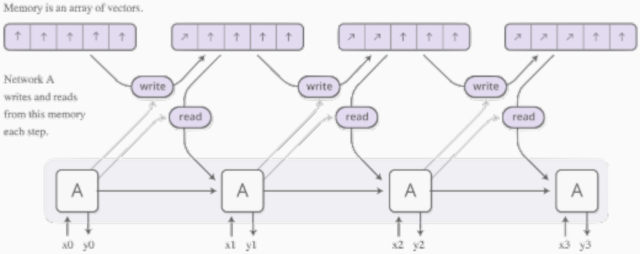


Graves et al. (2014). See also `https://distill.pub/2016/augmented-rnns/`

Neural Turing Machines

- Designed such that every component of the architecture is differentiable, making it straight forward to train with gradient descent.

- Controller: LSTM *or* Feedforward neural net.
- Reading head: Convex combination of the row-vectors M_t in memory: $r_t = W_t \otimes M_t$.

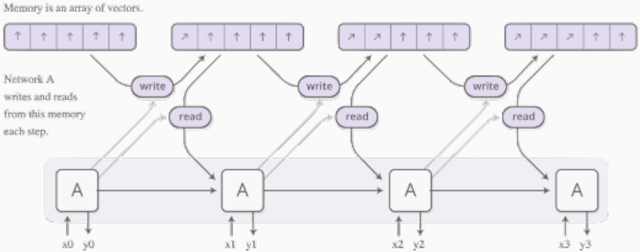


Graves et al. (2014). See also `https://distill.pub/2016/augmented-rnns/`

Neural Turing Machines

- Designed such that every component of the architecture is differentiable, making it straight forward to train with gradient descent.

- Controller: LSTM *or* Feedforward neural net.
- Reading head: Convex combination of the row-vectors M_t in memory: $r_t = W_t \otimes M_t$.
- Writting head: Two parts:
  1. *Erase* $\leftrightarrow$ Forget gate of LSTM. Given an *erase* vector, e_t: $\tilde{M}_t = M_{t-1}\,(\mathbb{1} - W_t e_t)$,
  2. *Add* $\leftrightarrow$ Input gate of LSTM. Given an *add* vector, a_t: $M_t = \tilde{M}_t + W_t a_t$.



Graves et al. (2014). See also `https://distill.pub/2016/augmented-rnns/`

Even Longer-Term Memory

Experiments: Copy a sequence of binary vectors, of size between 1 and 20.

Focus:

- Can NTM store and recall a long sequence of arbitrary information?
- And, in particular, is it able to overcome longer delays than LSTMs?

Even Longer-Term Memory

Experiments: Copy a sequence of binary vectors, of size between 1 and 20.

Focus:

- Can NTM store and recall a long sequence of arbitrary information?
- And, in particular, is it able to overcome longer delays than LSTMs?

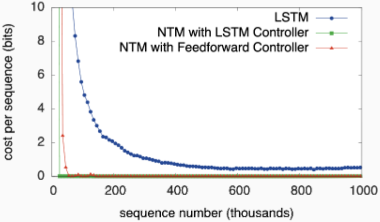


Figure 3: Copy Learning Curves.

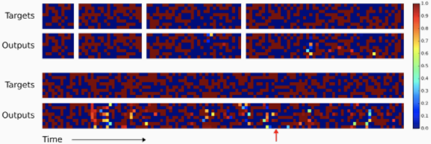


Figure 4: NTM Generalisation on the Copy Task.

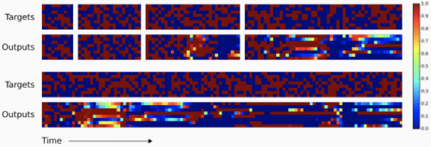


Figure 5: LSTM Generalisation on the Copy Task.

Graves, Wayne & Danihelka (2014)

But: How to know *where* to read and write? $\leadsto$ ***Attention mechanism***

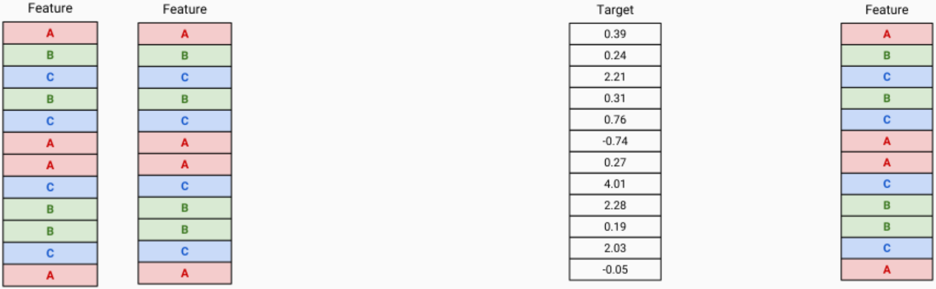
Appendix: Categorical Variable Encoding

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



Credit: Brendan Hasz blog post

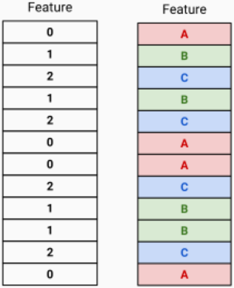
An others: Word2Vec, BERT (Bidirectional Encoder Representations from Transformers), . .

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



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
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
-0.03
0.76
2.25
0.76
2.25
-0.03
-0.03
2.25
0.76
0.76
2.25
-0.03

Credit: Brendan Hasz blog post

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

References i

Bahdanau, D., Cho, K., and Bengio, Y. (2014). **Neural machine translation by jointly learning to align and translate.** arXiv preprint arXiv:1409.0473.

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.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). **Empirical evaluation of gated recurrent neural networks on sequence modeling.** arXiv preprint arXiv:1412.3555.

El Hihi, S. and Bengio, Y. (1995). **Hierarchical recurrent neural networks for long-term dependencies.** In Nips, volume 409.

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

Gers, F. A. and Schmidhuber, J. (2000). **Recurrent nets that time and count.** In Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, volume 3, pages 189–194. IEEE.

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.

Graves, A., Wayne, G., and Danihelka, I. (2014). **Neural turing machines.** arXiv preprint arXiv:1410.5401.

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

Hochreiter, S. and Schmidhuber, J. (1997). **Long short-term memory.** Neural computation, 9(8):1735–1780.

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.

Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). **An empirical exploration of recurrent network architectures.** In International conference on machine learning, pages 2342–2350. PMLR.

Koutnik, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014). **A clockwork rnn.** In International Conference on Machine Learning, pages 1863–1871. PMLR.

Lin, T., Horne, B. G., Tino, P., and Giles, C. L. (1998). **Learning long-term dependencies is not as difficult with narx recurrent neural networks.** Technical report.

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).

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.

Mikolov, T. et al. (2012). **Statistical language models based on neural networks.** Presentation at Google, Mountain View, 2nd April, 80:26.

Mozer, M. C. (1991). **Induction of multiscale temporal structure.** Advances in neural information processing systems, 4.

Olah, C. **Understanding LSTM networks.** https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Olah, C. and Carter, S. **Attention and augmented recurrent neural networks.** https://distill.pub/2016/augmented-rnns/.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). **On the difficulty of training recurrent neural networks.** In International conference on machine learning, pages 1310–1318. PMLR.

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.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). **Attention is all you need.** Advances in neural information processing systems, 30.

Weston, J., Chopra, S., and Bordes, A. (2014). **Memory networks.** arXiv preprint arXiv:1410.3916.