CEE 616: Probabilistic Machine Learning
M3 Deep Neural Networks:
Lecture 3D: Neural Networks for Sequences

Jimi Oke

UMass Amherst
College of Engineering

Tue, Oct 28, 2025

Outline

Classical form of dynamical system

Classical form of dynamical system

A dynamical system is defined by the recurrent equation:

Classical form of dynamical system

A dynamical system is defined by the recurrent equation:

$$\boldsymbol{s}_t$$

Classical form of dynamical system

A dynamical system is defined by the recurrent equation:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{\theta}) \tag{1}$$

Classical form of dynamical system

A dynamical system is defined by the recurrent equation:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{s}_t$ is the state of the system and $\boldsymbol{\theta}$ are the parameters of f

Classical form of dynamical system

A dynamical system is defined by the recurrent equation:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{s}_t$ is the state of the system and $\boldsymbol{\theta}$ are the parameters of f

- recurrence: $\boldsymbol{s}_t$ is identically defined as $\boldsymbol{s}_{t-1}$

Classical form of dynamical system

A dynamical system is defined by the recurrent equation:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{s}_t$ is the state of the system and $\boldsymbol{\theta}$ are the parameters of f

- recurrence: $\boldsymbol{s}_t$ is identically defined as $\boldsymbol{s}_{t-1}$
- for τ time steps, definition is applied $\tau - 1$ times, e.g.

Classical form of dynamical system

A dynamical system is defined by the recurrent equation:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{s}_t$ is the state of the system and $\boldsymbol{\theta}$ are the parameters of f

- recurrence: $\boldsymbol{s}_t$ is identically defined as $\boldsymbol{s}_{t-1}$
- for τ time steps, definition is applied $\tau - 1$ times, e.g.

$$\boldsymbol{s}^{(3)} =$$

Classical form of dynamical system

A dynamical system is defined by the recurrent equation:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{s}_t$ is the state of the system and $\boldsymbol{\theta}$ are the parameters of f

- recurrence: $\boldsymbol{s}_t$ is identically defined as $\boldsymbol{s}_{t-1}$
- for τ time steps, definition is applied $\tau - 1$ times, e.g.

$$\boldsymbol{s}^{(3)} = f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta})$$

Classical form of dynamical system

A dynamical system is defined by the recurrent equation:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{s}_t$ is the state of the system and $\boldsymbol{\theta}$ are the parameters of f

- recurrence: $\boldsymbol{s}_t$ is identically defined as $\boldsymbol{s}_{t-1}$
- for τ time steps, definition is applied $\tau - 1$ times, e.g.

$$\boldsymbol{s}^{(3)} = f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) = f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta})$$

- writing out the recurrence relation in full yields an *unfolded* computational graph

Classical form of dynamical system

A dynamical system is defined by the recurrent equation:

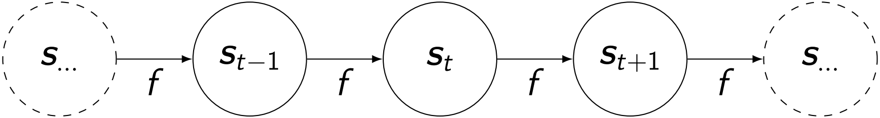
$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{s}_t$ is the state of the system and $\boldsymbol{\theta}$ are the parameters of f

- recurrence: $\boldsymbol{s}_t$ is identically defined as $\boldsymbol{s}_{t-1}$
- for τ time steps, definition is applied $\tau - 1$ times, e.g.

$$\boldsymbol{s}^{(3)} = f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) = f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta})$$

- writing out the recurrence relation in full yields an *unfolded* computational graph



Dynamical system driven by external signal

Dynamical system driven by external signal

The state $\boldsymbol{s}_t$ of a dynamical system driven by an external signal $\boldsymbol{x}_t$ can be described by

Dynamical system driven by external signal

The state $\boldsymbol{s}_t$ of a dynamical system driven by an external signal $\boldsymbol{x}_t$ can be described by

$$\boldsymbol{s}_t$$

Dynamical system driven by external signal

The state $\boldsymbol{s}_t$ of a dynamical system driven by an external signal $\boldsymbol{x}_t$ can be described by

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{x}_t; \boldsymbol{\theta}) \tag{2}$$

Dynamical system driven by external signal

The state $\boldsymbol{s}_t$ of a dynamical system driven by an external signal $\boldsymbol{x}_t$ can be described by

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{x}_t; \boldsymbol{\theta}) \tag{2}$$

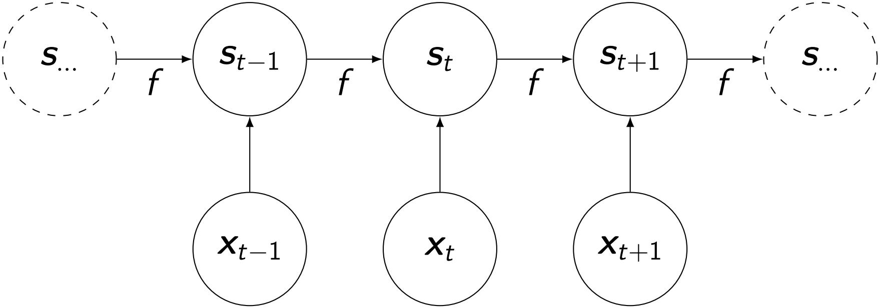
- the state $\boldsymbol{s}_t$ includes information about the entire past sequence
- the unfolded computational graph:

Dynamical system driven by external signal

The state $\boldsymbol{s}_t$ of a dynamical system driven by an external signal $\boldsymbol{x}_t$ can be described by

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{x}_t; \boldsymbol{\theta}) \tag{2}$$

- the state $\boldsymbol{s}_t$ includes information about the entire past sequence
- the unfolded computational graph:



Recurrent neural networks as sequence models

Recurrent neural networks as sequence models

- Dynamical systems are sequences (temporal, etc)

Recurrent neural networks as sequence models

- Dynamical systems are sequences (temporal, etc)
- Recurrence relations can be modeled by *recurrent* neural networks (RNNs)

Recurrent neural networks as sequence models

- Dynamical systems are sequences (temporal, etc)
- Recurrence relations can be modeled by *recurrent* neural networks (RNNs)
- In an RNN, hidden units $\boldsymbol{h}$ represent the state $\boldsymbol{s}$ of the system:

Recurrent neural networks as sequence models

- Dynamical systems are sequences (temporal, etc)
- Recurrence relations can be modeled by *recurrent* neural networks (RNNs)
- In an RNN, hidden units $\boldsymbol{h}$ represent the state $\boldsymbol{s}$ of the system:

$$\boldsymbol{h}_t =$$

Recurrent neural networks as sequence models

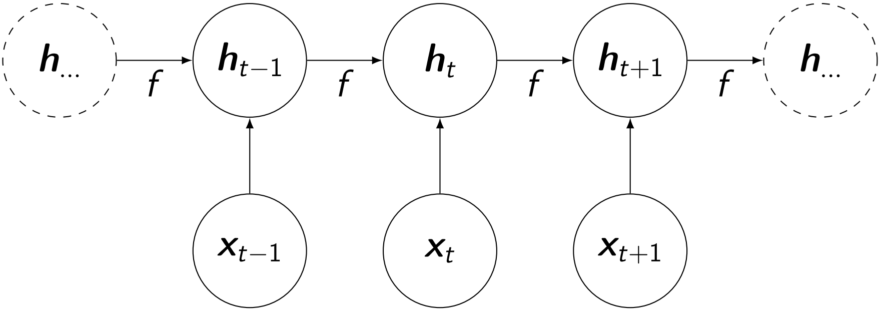
- Dynamical systems are sequences (temporal, etc)
- Recurrence relations can be modeled by *recurrent* neural networks (RNNs)
- In an RNN, hidden units $\boldsymbol{h}$ represent the state $\boldsymbol{s}$ of the system:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}) \tag{3}$$

Recurrent neural networks as sequence models

- Dynamical systems are sequences (temporal, etc)
- Recurrence relations can be modeled by *recurrent* neural networks (RNNs)
- In an RNN, hidden units $\boldsymbol{h}$ represent the state $\boldsymbol{s}$ of the system:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}) \tag{3}$$



Computational advantages of RNN

Computational advantages of RNN

- Transition function f maps past variable-length sequence $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ to fixed-length state $\boldsymbol{h}_t$

Computational advantages of RNN

- Transition function f maps past variable-length sequence $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ to fixed-length state $\boldsymbol{h}_t$
- Model always has same input size $\boldsymbol{h}_{t-1}$

Computational advantages of RNN

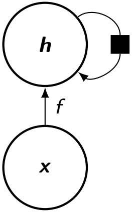
- Transition function f maps past variable-length sequence $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ to fixed-length state $\boldsymbol{h}_t$
- Model always has same input size $\boldsymbol{h}_{t-1}$
- Single function f with same parameters operates on all time steps

Computational advantages of RNN

- Transition function f maps past variable-length sequence $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ to fixed-length state $\boldsymbol{h}_t$
- Model always has same input size $\boldsymbol{h}_{t-1}$
- Single function f with same parameters operates on all time steps
- Further architectural features (including output layers) use information from $\boldsymbol{h}$ to make predictions

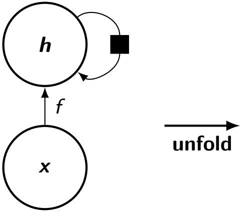
Computational advantages of RNN

- Transition function f maps past variable-length sequence $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ to fixed-length state $\boldsymbol{h}_t$
- Model always has same input size $\boldsymbol{h}_{t-1}$
- Single function f with same parameters operates on all time steps
- Further architectural features (including output layers) use information from $\boldsymbol{h}$ to make predictions



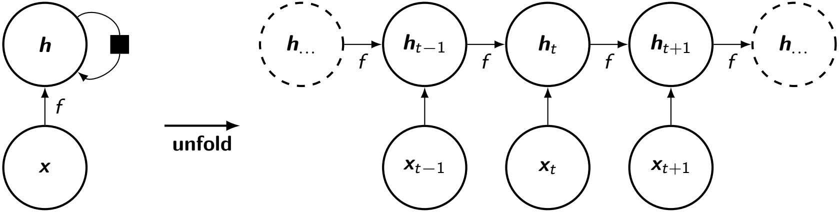
Computational advantages of RNN

- Transition function f maps past variable-length sequence $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ to fixed-length state $\boldsymbol{h}_t$
- Model always has same input size $\boldsymbol{h}_{t-1}$
- Single function f with same parameters operates on all time steps
- Further architectural features (including output layers) use information from $\boldsymbol{h}$ to make predictions



Computational advantages of RNN

- Transition function f maps past variable-length sequence $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ to fixed-length state $\boldsymbol{h}_t$
- Model always has same input size $\boldsymbol{h}_{t-1}$
- Single function f with same parameters operates on all time steps
- Further architectural features (including output layers) use information from $\boldsymbol{h}$ to make predictions



Computational advantages of RNN

- Transition function f maps past variable-length sequence $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ to fixed-length state $\boldsymbol{h}_t$
- Model always has same input size $\boldsymbol{h}_{t-1}$
- Single function f with same parameters operates on all time steps
- Further architectural features (including output layers) use information from $\boldsymbol{h}$ to make predictions

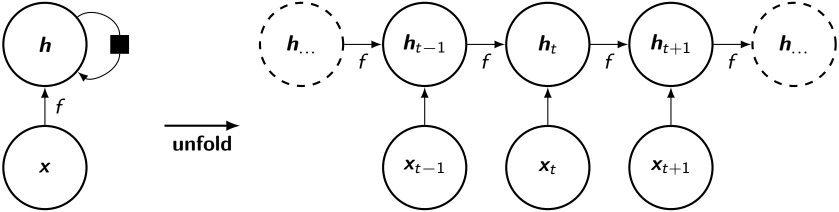


Figure: Recurrent network with no outputs. L: recurrent graph; R: time-unfolded computational graph. (Black square indicates time-step-delayed interaction)

RNN models for sequence tasks

RNN models for sequence tasks

- **Generation** (`Vec2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^{N_\infty C}$

RNN models for sequence tasks

- **Generation** (`Vec2Seq`): $f_{\theta} : \mathbb{R}^D \to \mathbb{R}^{N_\infty C}$
  - Input: vector of size D

RNN models for sequence tasks

- **Generation** (`Vec2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^D \rightarrow \mathbb{R}^{N_\infty C}$
  - Input: vector of size D
  - Output: arbitrary-length sequence of size C vectors

RNN models for sequence tasks

- **Generation** (`Vec2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^D \rightarrow \mathbb{R}^{N_\infty C}$
    - Input: vector of size D
    - Output: arbitrary-length sequence of size C vectors
    - Applications: language modeling, image captioning
- **Classification** (`Seq2Vec`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \rightarrow \mathbb{R}^C$

RNN models for sequence tasks

- **Generation** (`Vec2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^D \rightarrow \mathbb{R}^{N_\infty C}$
  - Input: vector of size D
  - Output: arbitrary-length sequence of size C vectors
  - Applications: language modeling, image captioning
- **Classification** (`Seq2Vec`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \rightarrow \mathbb{R}^C$
  - Input: variable-length sequence

RNN models for sequence tasks

- **Generation** (`Vec2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^{N_\infty C}$
  - Input: vector of size D
  - Output: arbitrary-length sequence of size C vectors
  - Applications: language modeling, image captioning
- **Classification** (`Seq2Vec`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \to \mathbb{R}^C$
  - Input: variable-length sequence
  - Output: fixed-length output vector (e.g. class label)

RNN models for sequence tasks

- **Generation** (`Vec2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{D} \rightarrow \mathbb{R}^{N_{\infty}C}$
  - Input: vector of size D
  - Output: arbitrary-length sequence of size C vectors
  - Applications: language modeling, image captioning
- **Classification** (`Seq2Vec`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \rightarrow \mathbb{R}^{C}$
  - Input: variable-length sequence
  - Output: fixed-length output vector (e.g. class label)
- **Translation** (`Seq2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \rightarrow \mathbb{R}^{T'C}$

RNN models for sequence tasks

- **Generation** (`Vec2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{D} \to \mathbb{R}^{N_{\infty} C}$
  - Input: vector of size D
  - Output: arbitrary-length sequence of size C vectors
  - Applications: language modeling, image captioning
- **Classification** (`Seq2Vec`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \to \mathbb{R}^{C}$
  - Input: variable-length sequence
  - Output: fixed-length output vector (e.g. class label)
- **Translation** (`Seq2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \to \mathbb{R}^{T'C}$
  - Aligned case: $T = T'$

RNN models for sequence tasks

- **Generation** (`Vec2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{D} \to \mathbb{R}^{N_{\infty} C}$
  - Input: vector of size D
  - Output: arbitrary-length sequence of size C vectors
  - Applications: language modeling, image captioning
- **Classification** (`Seq2Vec`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \to \mathbb{R}^{C}$
  - Input: variable-length sequence
  - Output: fixed-length output vector (e.g. class label)
- **Translation** (`Seq2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \to \mathbb{R}^{T'C}$
  - Aligned case: $T = T'$
  - Unaligned case: $T \neq T'$

RNN models for sequence tasks

- **Generation** (`Vec2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^{N_\infty C}$
  - Input: vector of size D
  - Output: arbitrary-length sequence of size C vectors
  - Applications: language modeling, image captioning
- **Classification** (`Seq2Vec`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \to \mathbb{R}^C$
  - Input: variable-length sequence
  - Output: fixed-length output vector (e.g. class label)
- **Translation** (`Seq2Seq`): $f_{\boldsymbol{\theta}} : \mathbb{R}^{TD} \to \mathbb{R}^{T'C}$
  - Aligned case: $T = T'$
  - Unaligned case: $T \neq T'$
  - Application: neural machine translation

Vec2Seq model for sequence generation

Vec2Seq model for sequence generation

Vec2Seq models map a fixed-length vector $\boldsymbol{x} \in \mathbb{R}^D$ onto a distribution over sequences $\boldsymbol{Y} \in \mathbb{R}^{T \times C}$ (or, $\boldsymbol{y}_{1:T} \in \boldsymbol{R}^C$; that is, T sequences, with each y_t a vector of length C)

Vec2Seq model for sequence generation

Vec2Seq models map a fixed-length vector $\boldsymbol{x} \in \mathbb{R}^D$ onto a distribution over sequences $\boldsymbol{Y} \in \mathbb{R}^{T \times C}$ (or, $\boldsymbol{y}_{1:T} \in \boldsymbol{R}^C$; that is, T sequences, with each y_t a vector of length C)

Thus, the generative model is given by:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) \tag{4}$$

Vec2Seq model for sequence generation

Vec2Seq models map a fixed-length vector $\boldsymbol{x} \in \mathbb{R}^D$ onto a distribution over sequences $\boldsymbol{Y} \in \mathbb{R}^{T \times C}$ (or, $\boldsymbol{y}_{1:T} \in \boldsymbol{R}^C$; that is, T sequences, with each y_t a vector of length C)

Thus, the generative model is given by:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) \tag{4}$$

where $\boldsymbol{h}_t$ is the hidden state of the model.

Vec2Seq model for sequence generation

Vec2Seq models map a fixed-length vector $\boldsymbol{x} \in \mathbb{R}^D$ onto a distribution over sequences $\boldsymbol{Y} \in \mathbb{R}^{T \times C}$ (or, $\boldsymbol{y}_{1:T} \in \boldsymbol{R}^C$; that is, T sequences, with each y_t a vector of length C)

Thus, the generative model is given by:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) \tag{4}$$

where $\boldsymbol{h}_t$ is the hidden state of the model.
By the multiplication rule, we can write:

Vec2Seq model for sequence generation

Vec2Seq models map a fixed-length vector $\boldsymbol{x} \in \mathbb{R}^D$ onto a distribution over sequences $\boldsymbol{Y} \in \mathbb{R}^{T \times C}$ (or, $\boldsymbol{y}_{1:T} \in \boldsymbol{R}^C$; that is, T sequences, with each y_t a vector of length C)

Thus, the generative model is given by:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) \tag{4}$$

where $\boldsymbol{h}_t$ is the hidden state of the model.
By the multiplication rule, we can write:

$$\sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x})$$

Vec2Seq model for sequence generation

Vec2Seq models map a fixed-length vector $\boldsymbol{x} \in \mathbb{R}^D$ onto a distribution over sequences $\boldsymbol{Y} \in \mathbb{R}^{T \times C}$ (or, $\boldsymbol{y}_{1:T} \in \boldsymbol{R}^C$; that is, T sequences, with each y_t a vector of length C)

Thus, the generative model is given by:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) \tag{4}$$

where $\boldsymbol{h}_t$ is the hidden state of the model.
By the multiplication rule, we can write:

$$\sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{h}_t) p(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{x}) \tag{5}$$

- Initial hidden state:

Vec2Seq model for sequence generation

Vec2Seq models map a fixed-length vector $\boldsymbol{x} \in \mathbb{R}^D$ onto a distribution over sequences $\boldsymbol{Y} \in \mathbb{R}^{T \times C}$ (or, $\boldsymbol{y}_{1:T} \in \boldsymbol{R}^C$; that is, T sequences, with each y_t a vector of length C)

Thus, the generative model is given by:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) \tag{4}$$

where $\boldsymbol{h}_t$ is the hidden state of the model.
By the multiplication rule, we can write:

$$\sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{h}_t) p(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{x}) \tag{5}$$

- Initial hidden state: $p(\boldsymbol{h}_1|\boldsymbol{h}_0, \boldsymbol{y}_0, \boldsymbol{x}) = p(\boldsymbol{h}_1|\boldsymbol{x})$

Vec2Seq model for sequence generation

Vec2Seq models map a fixed-length vector $\boldsymbol{x} \in \mathbb{R}^D$ onto a distribution over sequences $\boldsymbol{Y} \in \mathbb{R}^{T \times C}$ (or, $\boldsymbol{y}_{1:T} \in \boldsymbol{R}^C$; that is, T sequences, with each y_t a vector of length C)

Thus, the generative model is given by:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) \tag{4}$$

where $\boldsymbol{h}_t$ is the hidden state of the model.
By the multiplication rule, we can write:

$$\sum_{\boldsymbol{h}_{1:T}} p(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{x}) = \sum_{\boldsymbol{h}_{1:T}} \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{h}_t) p(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{x}) \tag{5}$$

- Initial hidden state: $p(\boldsymbol{h}_1|\boldsymbol{h}_0, \boldsymbol{y}_0, \boldsymbol{x}) = p(\boldsymbol{h}_1|\boldsymbol{x})$

Vec2Seq model (cont.)

Vec2Seq model (cont.)

The output distribution is given by:

$$p(\boldsymbol{y}_t|\boldsymbol{h}_t) = \begin{cases} \text{Cat}(\boldsymbol{y}_t|\mathcal{S}(\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o)), & \text{(qualitative outputs)} \\ \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o, \sigma^2\boldsymbol{I}), & \text{(real-valued outputs)} \end{cases} \tag{6}$$

Vec2Seq model (cont.)

The output distribution is given by:

$$p(\boldsymbol{y}_t|\boldsymbol{h}_t) = \begin{cases} \mathrm{Cat}(\boldsymbol{y}_t|\mathcal{S}(\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o)), & \text{(qualitative outputs)} \\ \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o, \sigma^2\boldsymbol{I}), & \text{(real-valued outputs)} \end{cases} \tag{6}$$

- $\boldsymbol{W}_{ho}$: matrix of hidden–output weights

Vec2Seq model (cont.)

The output distribution is given by:

$$p(\boldsymbol{y}_t|\boldsymbol{h}_t) = \begin{cases} \mathrm{Cat}(\boldsymbol{y}_t|\mathcal{S}(\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o)), & \text{(qualitative outputs)} \\ \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o, \sigma^2\boldsymbol{I}), & \text{(real-valued outputs)} \end{cases} \tag{6}$$

- $\boldsymbol{W}_{ho}$: matrix of hidden–output weights
- $\boldsymbol{b}_o$: bias term

Vec2Seq model (cont.)

The output distribution is given by:

$$p(\boldsymbol{y}_t|\boldsymbol{h}_t) = \begin{cases} \text{Cat}(\boldsymbol{y}_t|\mathcal{S}(\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o)), & \text{(qualitative outputs)} \\ \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o, \sigma^2\boldsymbol{I}), & \text{(real-valued outputs)} \end{cases} \tag{6}$$

- $\boldsymbol{W}_{ho}$: matrix of hidden–output weights
- $\boldsymbol{b}_o$: bias term
- $\boldsymbol{y}_t$ is the observed vector, while $\boldsymbol{o}_t$ is the predicted value (using NN notation)

Vec2Seq model (cont.)

The output distribution is given by:

$$p(\boldsymbol{y}_t|\boldsymbol{h}_t) = \begin{cases} \text{Cat}(\boldsymbol{y}_t|\mathcal{S}(\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o)), & \text{(qualitative outputs)} \\ \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o, \sigma^2\boldsymbol{I}), & \text{(real-valued outputs)} \end{cases} \tag{6}$$

- $\boldsymbol{W}_{ho}$: matrix of hidden–output weights
- $\boldsymbol{b}_o$: bias term
- $\boldsymbol{y}_t$ is the observed vector, while $\boldsymbol{o}_t$ is the predicted value (using NN notation)

The hidden state is typically deterministic:

Vec2Seq model (cont.)

The output distribution is given by:

$$p(\boldsymbol{y}_t|\boldsymbol{h}_t) = \begin{cases} \text{Cat}(\boldsymbol{y}_t|\mathcal{S}(\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o)), & \text{(qualitative outputs)} \\ \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o, \sigma^2\boldsymbol{I}), & \text{(real-valued outputs)} \end{cases} \tag{6}$$

- $\boldsymbol{W}_{ho}$: matrix of hidden–output weights
- $\boldsymbol{b}_o$: bias term
- $\boldsymbol{y}_t$ is the observed vector, while $\boldsymbol{o}_t$ is the predicted value (using NN notation)

The hidden state is typically deterministic:

$$\boldsymbol{h}_t = \varphi(\boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \tag{7}$$

Vec2Seq model (cont.)

The output distribution is given by:

$$p(\boldsymbol{y}_t|\boldsymbol{h}_t) = \begin{cases} \text{Cat}(\boldsymbol{y}_t|\mathcal{S}(\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o)), & \text{(qualitative outputs)} \\ \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o, \sigma^2\boldsymbol{I}), & \text{(real-valued outputs)} \end{cases} \tag{6}$$

- $\boldsymbol{W}_{ho}$: matrix of hidden–output weights
- $\boldsymbol{b}_o$: bias term
- $\boldsymbol{y}_t$ is the observed vector, while $\boldsymbol{o}_t$ is the predicted value (using NN notation)

The hidden state is typically deterministic:

$$\boldsymbol{h}_t = \varphi(\boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \tag{7}$$

- $\boldsymbol{W}_{xh}$: input–hidden weight matrix

Vec2Seq model (cont.)

The output distribution is given by:

$$p(\boldsymbol{y}_t|\boldsymbol{h}_t) = \begin{cases} \text{Cat}(\boldsymbol{y}_t|\mathcal{S}(\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o)), & \text{(qualitative outputs)} \\ \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o, \sigma^2\boldsymbol{I}), & \text{(real-valued outputs)} \end{cases} \tag{6}$$

- $\boldsymbol{W}_{ho}$: matrix of hidden–output weights
- $\boldsymbol{b}_o$: bias term
- $\boldsymbol{y}_t$ is the observed vector, while $\boldsymbol{o}_t$ is the predicted value (using NN notation)

The hidden state is typically deterministic:

$$\boldsymbol{h}_t = \varphi(\boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \tag{7}$$

- $\boldsymbol{W}_{xh}$: input–hidden weight matrix
- $\boldsymbol{W}_{hh}$: hidden–hidden weight matrix

Vec2Seq model (cont.)

The output distribution is given by:

$$p(\boldsymbol{y}_t|\boldsymbol{h}_t) = \begin{cases} \mathrm{Cat}(\boldsymbol{y}_t|\mathcal{S}(\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o)), & \text{(qualitative outputs)} \\ \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o, \sigma^2\boldsymbol{I}), & \text{(real-valued outputs)} \end{cases} \tag{6}$$

- $\boldsymbol{W}_{ho}$: matrix of hidden–output weights
- $\boldsymbol{b}_o$: bias term
- $\boldsymbol{y}_t$ is the observed vector, while $\boldsymbol{o}_t$ is the predicted value (using NN notation)

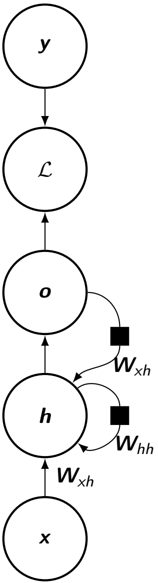
The hidden state is typically deterministic:

$$\boldsymbol{h}_t = \varphi(\boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \tag{7}$$

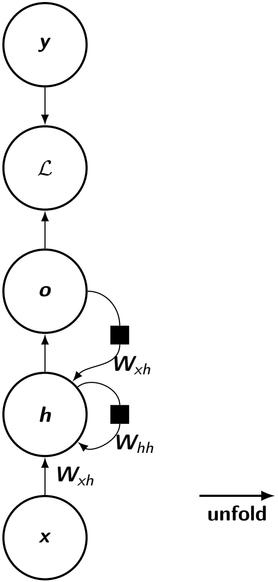
- $\boldsymbol{W}_{xh}$: input–hidden weight matrix
- $\boldsymbol{W}_{hh}$: hidden–hidden weight matrix
- $\boldsymbol{b}_h$: bias term

Vec2Seq RNN: circuit diagram and computation graph

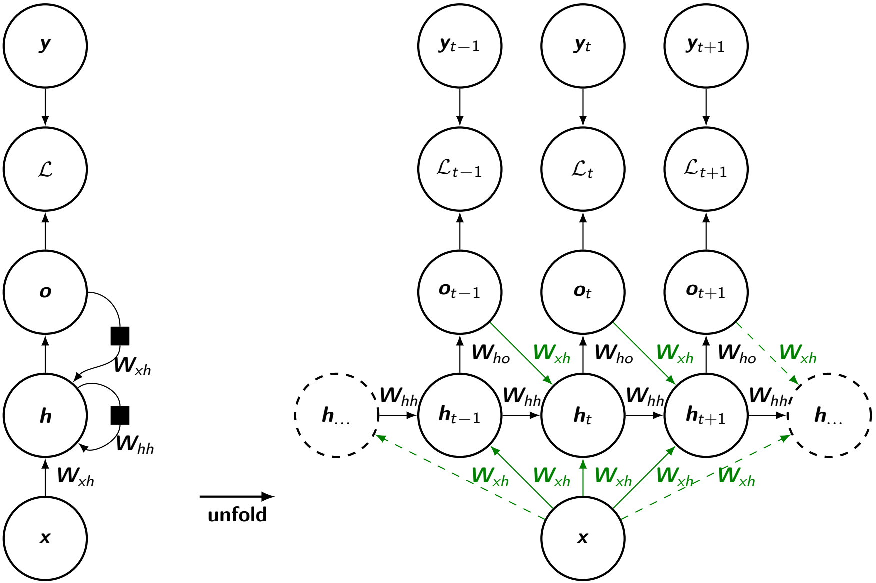
Vec2Seq RNN: circuit diagram and computation graph



Vec2Seq RNN: circuit diagram and computation graph

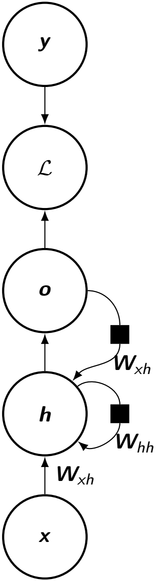


Vec2Seq RNN: circuit diagram and computation graph

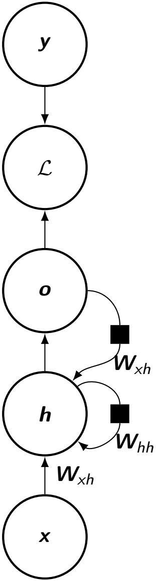


Vec2Seq model summary

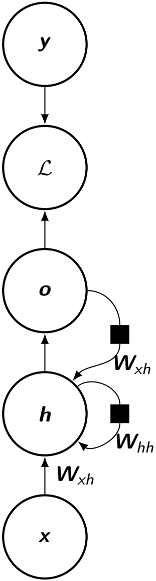
Vec2Seq model summary



Vec2Seq model summary



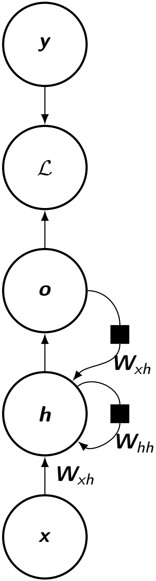
Vec2Seq model summary



Update equations:

$$\boldsymbol{a}_t = \boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h \tag{8}$$

Vec2Seq model summary

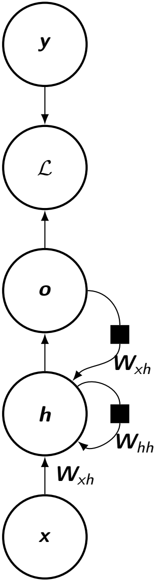


Update equations:

$$\boldsymbol{a}_t = \boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h \quad (8)$$

$$\boldsymbol{h}_t = \varphi(\boldsymbol{a}_t) \quad (9)$$

Vec2Seq model summary



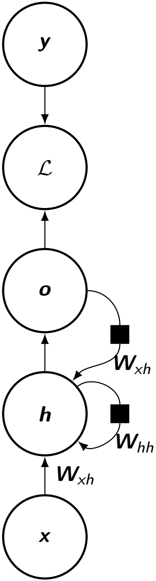
Update equations:

$$\boldsymbol{a}_t = \boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h \tag{8}$$

$$\boldsymbol{h}_t = \varphi(\boldsymbol{a}_t) \tag{9}$$

$$\boldsymbol{o}_t =$$

Vec2Seq model summary



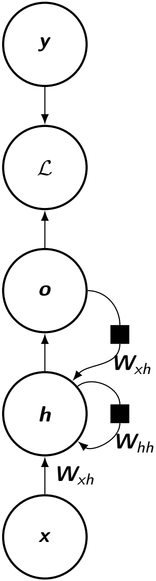
Update equations:

$$\boldsymbol{a}_t = \boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h \tag{8}$$

$$\boldsymbol{h}_t = \varphi(\boldsymbol{a}_t) \tag{9}$$

$$\boldsymbol{o}_t = \boldsymbol{W}_{oh}\boldsymbol{h}_t + \boldsymbol{b}_o \tag{10}$$

Vec2Seq model summary



Update equations:

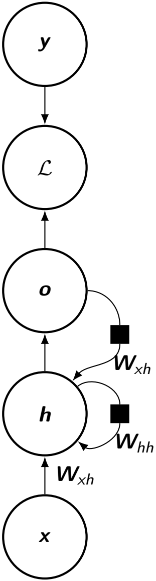
$$\boldsymbol{a}_t = \boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h \tag{8}$$

$$\boldsymbol{h}_t = \varphi(\boldsymbol{a}_t) \tag{9}$$

$$\boldsymbol{o}_t = \boldsymbol{W}_{oh}\boldsymbol{h}_t + \boldsymbol{b}_o \tag{10}$$

$$\hat{\boldsymbol{y}}_t = \mathcal{S}(\boldsymbol{o}_t) \tag{11}$$

Vec2Seq model summary



Update equations:

$$\boldsymbol{a}_t = \boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h \tag{8}$$

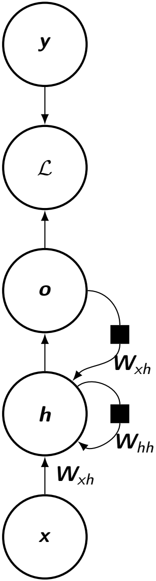
$$\boldsymbol{h}_t = \varphi(\boldsymbol{a}_t) \tag{9}$$

$$\boldsymbol{o}_t = \boldsymbol{W}_{oh}\boldsymbol{h}_t + \boldsymbol{b}_o \tag{10}$$

$$\hat{\boldsymbol{y}}_t = \mathcal{S}(\boldsymbol{o}_t) \tag{11}$$

- In typical applications, $\boldsymbol{o}_t = [o_{t1}, o_{t2}, \ldots, o_{tC}]$ (e.g. one-hot vector, each representing a character)

Vec2Seq model summary

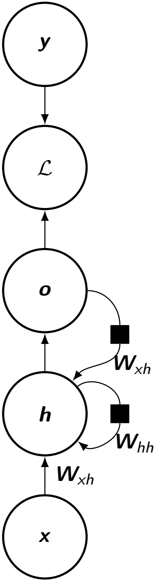


Update equations:

$$\begin{aligned} \boldsymbol{a}_t &= \boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h & (8) \\ \boldsymbol{h}_t &= \varphi(\boldsymbol{a}_t) & (9) \\ \boldsymbol{o}_t &= \boldsymbol{W}_{oh}\boldsymbol{h}_t + \boldsymbol{b}_o & (10) \\ \hat{\boldsymbol{y}}_t &= \mathcal{S}(\boldsymbol{o}_t) & (11) \end{aligned}$$

- In typical applications, $\boldsymbol{o}_t = [o_{t1}, o_{t2}, \ldots, o_{tC}]$ (e.g. one-hot vector, each representing a character)
- $\boldsymbol{o}_t$ depends on $\boldsymbol{h}_t$, which depends on $\boldsymbol{o}_{t-1}$

Vec2Seq model summary

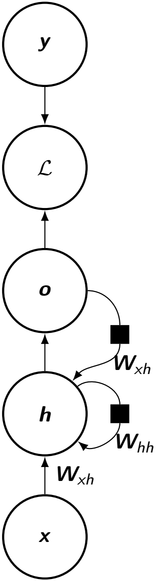


Update equations:

$$\begin{aligned}
\boldsymbol{a}_t &= \boldsymbol{W}_{xh}[\boldsymbol{x};\boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h && (8)\\
\boldsymbol{h}_t &= \varphi(\boldsymbol{a}_t) && (9)\\
\boldsymbol{o}_t &= \boldsymbol{W}_{oh}\boldsymbol{h}_t + \boldsymbol{b}_o && (10)\\
\hat{\boldsymbol{y}}_t &= \mathcal{S}(\boldsymbol{o}_t) && (11)
\end{aligned}$$

- In typical applications, $\boldsymbol{o}_t = [o_{t1}, o_{t2}, \ldots, o_{tC}]$ (e.g. one-hot vector, each representing a character)
- $\boldsymbol{o}_t$ depends on $\boldsymbol{h}_t$, which depends on $\boldsymbol{o}_{t-1}$
- $\boldsymbol{o}_t$ depends on *all* past observations and the fixed input $\boldsymbol{x}$

Vec2Seq model summary

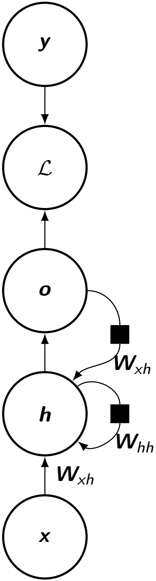


Update equations:

$$
\begin{aligned}
\boldsymbol{a}_t &= \boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h & (8)\\
\boldsymbol{h}_t &= \varphi(\boldsymbol{a}_t) & (9)\\
\boldsymbol{o}_t &= \boldsymbol{W}_{oh}\boldsymbol{h}_t + \boldsymbol{b}_o & (10)\\
\hat{\boldsymbol{y}}_t &= \mathcal{S}(\boldsymbol{o}_t) & (11)
\end{aligned}
$$

- In typical applications, $\boldsymbol{o}_t = [o_{t1}, o_{t2}, \ldots, o_{tC}]$ (e.g. one-hot vector, each representing a character)
- $\boldsymbol{o}_t$ depends on $\boldsymbol{h}_t$, which depends on $\boldsymbol{o}_{t-1}$
- $\boldsymbol{o}_t$ depends on *all* past observations and the fixed input $\boldsymbol{x}$
- $[\boldsymbol{x}; \boldsymbol{o}_{t-1}]$ denotes the stacking of $\boldsymbol{x}$ and $\boldsymbol{o}_{t-1}$

Vec2Seq model summary

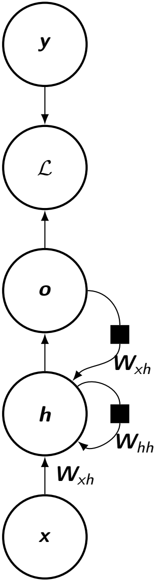


Update equations:

$$\begin{aligned} \boldsymbol{a}_t &= \boldsymbol{W}_{xh}[\boldsymbol{x}; \boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h \quad (8) \\ \boldsymbol{h}_t &= \varphi(\boldsymbol{a}_t) \quad (9) \\ \boldsymbol{o}_t &= \boldsymbol{W}_{oh}\boldsymbol{h}_t + \boldsymbol{b}_o \quad (10) \\ \hat{\boldsymbol{y}}_t &= \mathcal{S}(\boldsymbol{o}_t) \quad (11) \end{aligned}$$

- In typical applications, $\boldsymbol{o}_t = [o_{t1}, o_{t2}, \ldots, o_{tC}]$ (e.g. one-hot vector, each representing a character)
- $\boldsymbol{o}_t$ depends on $\boldsymbol{h}_t$, which depends on $\boldsymbol{o}_{t-1}$
- $\boldsymbol{o}_t$ depends on *all* past observations and the fixed input $\boldsymbol{x}$
- $[\boldsymbol{x}; \boldsymbol{o}_{t-1}]$ denotes the stacking of $\boldsymbol{x}$ and $\boldsymbol{o}_{t-1}$
- So, if $\boldsymbol{x} \in \mathbb{R}^D$, $\boldsymbol{o}_t \in \mathbb{R}^C$, and $\boldsymbol{h}_t \in \mathbb{R}^M$, then $\boldsymbol{W}_{xh} \in \mathbb{R}^{(D+C)\times M}$

Vec2Seq model summary



Update equations:

$$\begin{aligned}
\boldsymbol{a}_t &= \boldsymbol{W}_{xh}[\boldsymbol{x};\boldsymbol{o}_{t-1}] + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h && (8)\\
\boldsymbol{h}_t &= \varphi(\boldsymbol{a}_t) && (9)\\
\boldsymbol{o}_t &= \boldsymbol{W}_{oh}\boldsymbol{h}_t + \boldsymbol{b}_o && (10)\\
\hat{\boldsymbol{y}}_t &= \mathcal{S}(\boldsymbol{o}_t) && (11)
\end{aligned}$$

- In typical applications, $\boldsymbol{o}_t = [o_{t1}, o_{t2}, \ldots, o_{tC}]$ (e.g. one-hot vector, each representing a character)
- $\boldsymbol{o}_t$ depends on $\boldsymbol{h}_t$, which depends on $\boldsymbol{o}_{t-1}$
- $\boldsymbol{o}_t$ depends on *all* past observations and the fixed input $\boldsymbol{x}$
- $[\boldsymbol{x};\boldsymbol{o}_{t-1}]$ denotes the stacking of $\boldsymbol{x}$ and $\boldsymbol{o}_{t-1}$
- So, if $\boldsymbol{x} \in \mathbb{R}^D$, $\boldsymbol{o}_t \in \mathbb{R}^C$, and $\boldsymbol{h}_t \in \mathbb{R}^M$, then $\boldsymbol{W}_{xh} \in \mathbb{R}^{(D+C)\times M}$
- M is the number of units (neurons) in the hidden layer

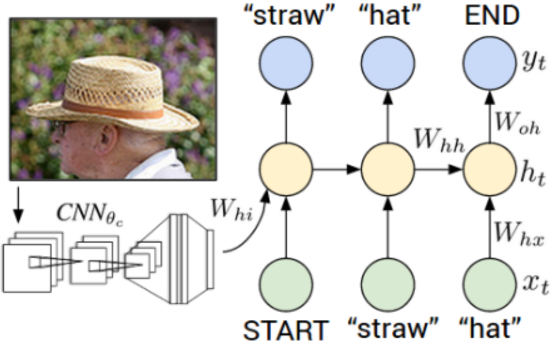
Application: image captioning

Application: image captioning

Image or processed version from CNN is used as input, with output as sequence of descriptive words

Application: image captioning

Image or processed version from CNN is used as input, with output as sequence of descriptive words



Source: https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2

Seq2Seq model: aligned case

Seq2Seq model: aligned case

Seq2seq models map a sequence of vectors $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto another sequence $\boldsymbol{y}_{1:T'} \in \mathbb{R}^C$.

Seq2Seq model: aligned case

Seq2seq models map a sequence of vectors $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto another sequence $\boldsymbol{y}_{1:T'} \in \mathbb{R}^C$. We consider the aligned case (dense sequence modeling) where $T = T'$:

$$p(\boldsymbol{y}_{1:T} | \boldsymbol{x}_{1:T}) =$$

Seq2Seq model: aligned case

Seq2seq models map a sequence of vectors $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto another sequence $\boldsymbol{y}_{1:T'} \in \mathbb{R}^C$. We consider the aligned case (dense sequence modeling) where $T = T'$:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T}) = \sum_{\boldsymbol{h}_{1:T}} \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{h}_t)\mathbb{I}(\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)) \tag{12}$$

Seq2Seq model: aligned case

Seq2seq models map a sequence of vectors $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto another sequence $\boldsymbol{y}_{1:T'} \in \mathbb{R}^C$. We consider the aligned case (dense sequence modeling) where $T = T'$:

$$p(\boldsymbol{y}_{1:T} | \boldsymbol{x}_{1:T}) = \sum_{\boldsymbol{h}_{1:T}} \prod_{t=1}^{T} p(\boldsymbol{y}_t | \boldsymbol{h}_t) \mathbb{I}(\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)) \tag{12}$$

where the initial hidden state is $\boldsymbol{h}_1 = f(\boldsymbol{h}_0, \boldsymbol{x}_1) = f_0(\boldsymbol{x}_1)$.

Seq2Seq model: aligned case

Seq2seq models map a sequence of vectors $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto another sequence $\boldsymbol{y}_{1:T'} \in \mathbb{R}^C$. We consider the aligned case (dense sequence modeling) where $T = T'$:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T}) = \sum_{\boldsymbol{h}_{1:T}} \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{h}_t)\mathbb{I}(\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)) \tag{12}$$

where the initial hidden state is $\boldsymbol{h}_1 = f(\boldsymbol{h}_0, \boldsymbol{x}_1) = f_0(\boldsymbol{x}_1)$.

- The hidden state is given by:

Seq2Seq model: aligned case

Seq2seq models map a sequence of vectors $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto another sequence $\boldsymbol{y}_{1:T'} \in \mathbb{R}^C$. We consider the aligned case (dense sequence modeling) where $T = T'$:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T}) = \sum_{\boldsymbol{h}_{1:T}} \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{h}_t)\mathbb{I}(\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)) \tag{12}$$

where the initial hidden state is $\boldsymbol{h}_1 = f(\boldsymbol{h}_0, \boldsymbol{x}_1) = f_0(\boldsymbol{x}_1)$.

- The hidden state is given by:

$$\boldsymbol{h}_t = \varphi(\boldsymbol{W}_{xh}\boldsymbol{h}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{h}) \tag{13}$$

Seq2Seq model: aligned case

Seq2seq models map a sequence of vectors $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto another sequence $\boldsymbol{y}_{1:T'} \in \mathbb{R}^C$. We consider the aligned case (dense sequence modeling) where $T = T'$:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T}) = \sum_{\boldsymbol{h}_{1:T}} \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{h}_t)\mathbb{I}(\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)) \tag{12}$$

where the initial hidden state is $\boldsymbol{h}_1 = f(\boldsymbol{h}_0, \boldsymbol{x}_1) = f_0(\boldsymbol{x}_1)$.

- The hidden state is given by:

$$\boldsymbol{h}_t = \varphi(\boldsymbol{W}_{xh}\boldsymbol{h}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{h}) \tag{13}$$

- The output is given by:

Seq2Seq model: aligned case

Seq2seq models map a sequence of vectors $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto another sequence $\boldsymbol{y}_{1:T'} \in \mathbb{R}^C$. We consider the aligned case (dense sequence modeling) where $T = T'$:

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T}) = \sum_{\boldsymbol{h}_{1:T}} \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{h}_t)\mathbb{I}(\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)) \tag{12}$$

where the initial hidden state is $\boldsymbol{h}_1 = f(\boldsymbol{h}_0, \boldsymbol{x}_1) = f_0(\boldsymbol{x}_1)$.

- The hidden state is given by:

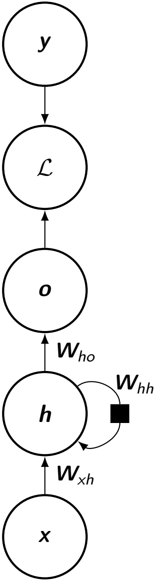
$$\boldsymbol{h}_t = \varphi(\boldsymbol{W}_{xh}\boldsymbol{h}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{h}) \tag{13}$$

- The output is given by:

$$\boldsymbol{o}_t = \boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o \tag{14}$$

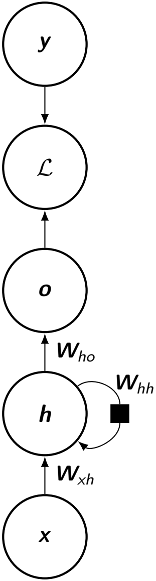
Aligned seq2seq circuit diagram

Aligned seq2seq circuit diagram



The update equations for a seq2seq RNN are given by

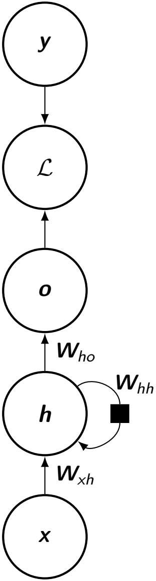
Aligned seq2seq circuit diagram



The update equations for a seq2seq RNN are given by

$$\boldsymbol{a}_t = \boldsymbol{b}_h + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t \tag{15}$$

Aligned seq2seq circuit diagram

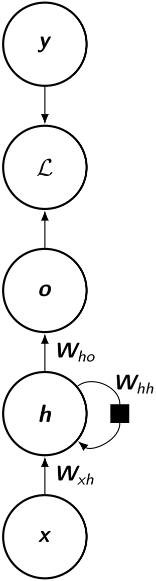


The update equations for a seq2seq RNN are given by

$$\boldsymbol{a}_t = \boldsymbol{b}_h + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t \tag{15}$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{a}_t) \tag{16}$$

Aligned seq2seq circuit diagram



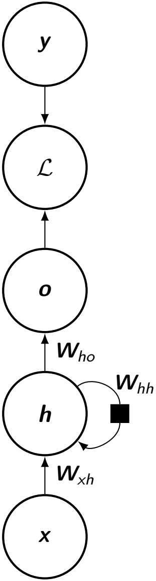
The update equations for a seq2seq RNN are given by

$$\boldsymbol{a}_t = \boldsymbol{b}_h + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t \tag{15}$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{a}_t) \tag{16}$$

$$\boldsymbol{o}_t = \boldsymbol{b}_o + \boldsymbol{W}_{ho}\boldsymbol{h}_t \tag{17}$$

Aligned seq2seq circuit diagram



The update equations for a seq2seq RNN are given by

$$\boldsymbol{a}_t = \boldsymbol{b}_h + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t \tag{15}$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{a}_t) \tag{16}$$

$$\boldsymbol{o}_t = \boldsymbol{b}_o + \boldsymbol{W}_{ho}\boldsymbol{h}_t \tag{17}$$

- $\boldsymbol{x}$: input sequence
- $\boldsymbol{h}$: hidden units
- $\boldsymbol{b}_h$, $\boldsymbol{b}_o$: bias vectors
- $\boldsymbol{W}_{xh}$: weight matrix of input–hidden unit connections
- $\boldsymbol{W}_{hh}$: weight matrix of hidden–hidden unit connections
- $\boldsymbol{W}_{ho}$: weight matrix of hidden–output unit connections
- $\boldsymbol{o}$: output vector
- $\boldsymbol{y}$ target sequence
- $\mathcal{L}$: loss function measuring error between $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$

Seq2seq model: unaligned case

Seq2seq model: unaligned case

To map a sequence of length T to another of length T', we use an **encoder-decoder architecture**:

Seq2seq model: unaligned case

To map a sequence of length T to another of length T', we use an **encoder-decoder architecture**:

- The encoder f_e maps the input sequence onto a context vector

$$\boldsymbol{c} = f_e(\boldsymbol{x}_{1:T}) \tag{18}$$

Seq2seq model: unaligned case

To map a sequence of length T to another of length T', we use an **encoder-decoder architecture**:

- The encoder f_e maps the input sequence onto a context vector

$$\boldsymbol{c} = f_e(\boldsymbol{x}_{1:T}) \tag{18}$$

- The decoder f_d generates the output sequence by mapping from the context vector:

Seq2seq model: unaligned case

To map a sequence of length T to another of length T', we use an **encoder-decoder architecture**:

- The encoder f_e maps the input sequence onto a context vector

$$\boldsymbol{c} = f_e(\boldsymbol{x}_{1:T}) \tag{18}$$

- The decoder f_d generates the output sequence by mapping from the context vector:

$$\boldsymbol{y}_{1:T'} = f_d(\boldsymbol{c}) \tag{19}$$

Seq2seq model: unaligned case

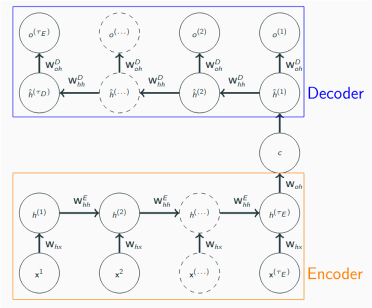
To map a sequence of length T to another of length T', we use an **encoder-decoder architecture**:

- The encoder f_e maps the input sequence onto a context vector

$$\boldsymbol{c} = f_e(\boldsymbol{x}_{1:T}) \qquad (18)$$

- The decoder f_d generates the output sequence by mapping from the context vector:

$$\boldsymbol{y}_{1:T'} = f_d(\boldsymbol{c}) \qquad (19)$$



Source: https://www.inf.ed.ac.uk/teaching/courses/mlp/2019-20/lectures/mlp09-rnn.pdf

Illustration of seq2seq for translation

Illustration of seq2seq for translation

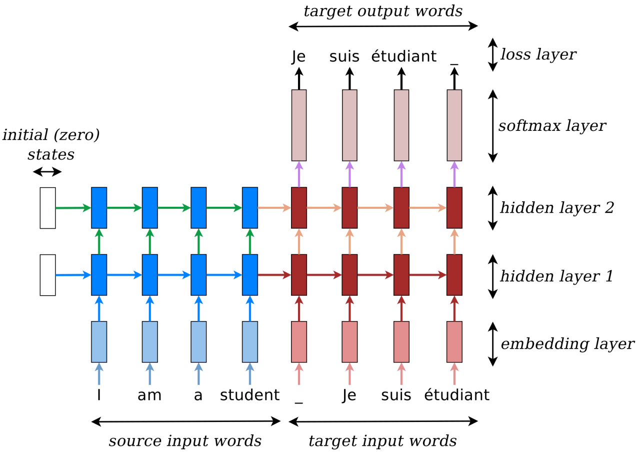


Illustration of seq2seq model for English-to-French translation.

Source: `https://github.com/probml/pml-book/blob/main/book1-figures/Figure_15.8_A.png`

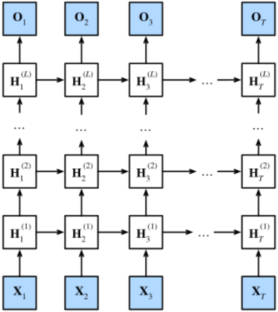
Deep RNNs

Deep RNNs

More complex models can be developed by stacking **hidden chains**.

Deep RNNs

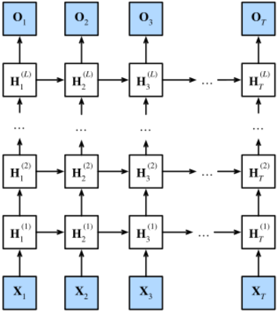
More complex models can be developed by stacking **hidden chains**.



Source: `https://d2l.ai/chapter_recurrent-modern/deep-rnn.html`

Deep RNNs

More complex models can be developed by stacking **hidden chains**.

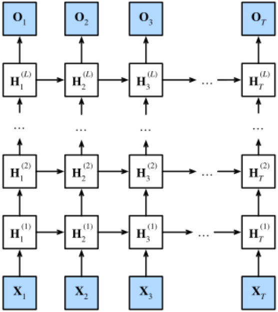


Source: `https://d2l.ai/chapter_recurrent-modern/deep-rnn.html`

The hidden state for layer ℓ at time t is then given by:

Deep RNNs

More complex models can be developed by stacking **hidden chains**.



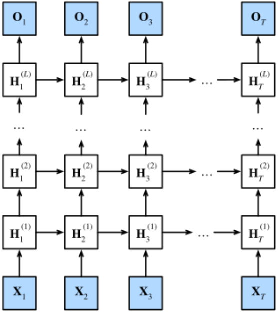
Source: `https://d2l.ai/chapter_recurrent-modern/deep-rnn.html`

The hidden state for layer ℓ at time t is then given by:

$$\boldsymbol{h}_t^{\ell} = \varphi(\boldsymbol{W}_{xh}^{\ell}\boldsymbol{h}_t^{\ell-1} + \boldsymbol{W}_{hh}^{\ell}\boldsymbol{h}_{t-1}^{\ell} + \boldsymbol{h}^{\ell}) \tag{20}$$

Deep RNNs

More complex models can be developed by stacking **hidden chains**.



Source: `https://d2l.ai/chapter_recurrent-modern/deep-rnn.html`

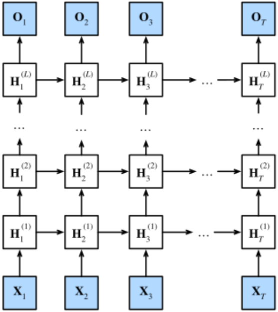
The hidden state for layer ℓ at time t is then given by:

$$\boldsymbol{h}_t^{\ell} = \varphi(\boldsymbol{W}_{xh}^{\ell}\boldsymbol{h}_t^{\ell-1} + \boldsymbol{W}_{hh}^{\ell}\boldsymbol{h}_{t-1}^{\ell} + \boldsymbol{h}^{\ell}) \tag{20}$$

And the output at each time step as:

Deep RNNs

More complex models can be developed by stacking **hidden chains**.



Source: https://d2l.ai/chapter_recurrent-modern/deep-rnn.html

The hidden state for layer ℓ at time t is then given by:

$$\boldsymbol{h}_t^\ell = \varphi(\boldsymbol{W}_{xh}^\ell \boldsymbol{h}_t^{\ell-1} + \boldsymbol{W}_{hh}^\ell \boldsymbol{h}_{t-1}^\ell + \boldsymbol{h}^\ell) \tag{20}$$

And the output at each time step as:

$$\boldsymbol{o}_t = \boldsymbol{W}_{ho} \boldsymbol{h}_t^L + \boldsymbol{b}_o \tag{21}$$

Seq2vec models for sequence classification

Seq2vec models for sequence classification

Seq2vec models map a sequence $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto a fixed length vector $\boldsymbol{y} \in \mathbb{R}^C$ (e.g. class label)

Seq2vec models for sequence classification

Seq2vec models map a sequence $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto a fixed length vector $\boldsymbol{y} \in \mathbb{R}^C$ (e.g. class label)

In the simple approach, the output depends on final state only.

Seq2vec models for sequence classification

Seq2vec models map a sequence $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto a fixed length vector $\boldsymbol{y} \in \mathbb{R}^C$ (e.g. class label)

In the simple approach, the output depends on final state only.

Thus, the model can be specified as:

Seq2vec models for sequence classification

Seq2vec models map a sequence $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto a fixed length vector $\boldsymbol{y} \in \mathbb{R}^C$ (e.g. class label)

In the simple approach, the output depends on final state only.

Thus, the model can be specified as:

$$p(y|\boldsymbol{x}_{1:T}) = \mathrm{Cat}(y|\mathcal{S}(\boldsymbol{W}\boldsymbol{h}_T) \quad (22)$$

Seq2vec models for sequence classification

Seq2vec models map a sequence $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto a fixed length vector $\boldsymbol{y} \in \mathbb{R}^C$ (e.g. class label)

In the simple approach, the output depends on final state only.

Thus, the model can be specified as:

$$p(y|\boldsymbol{x}_{1:T}) = \mathrm{Cat}(y|\mathcal{S}(\boldsymbol{W}\boldsymbol{h}_T)) \tag{22}$$

where $\boldsymbol{h}_T$ is the final state of the RNN

Seq2vec models for sequence classification

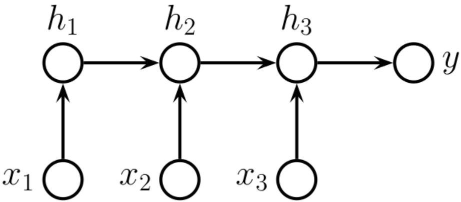
Seq2vec models map a sequence $\boldsymbol{x}_{1:T} \in \mathbb{R}^D$ onto a fixed length vector $\boldsymbol{y} \in \mathbb{R}^C$ (e.g. class label)

In the simple approach, the output depends on final state only.

Thus, the model can be specified as:

$$p(y|\boldsymbol{x}_{1:T}) = \text{Cat}(y|\mathcal{S}(\boldsymbol{W}\boldsymbol{h}_T)) \tag{22}$$

where $\boldsymbol{h}_T$ is the final state of the RNN



Bidirectional seq2vec model

Bidirectional models allow the output to depend on the entire sequence (i.e. the hidden state depends on past and future contexts):

Bidirectional seq2vec model

Bidirectional models allow the output to depend on the entire sequence (i.e. the hidden state depends on past and future contexts):

$$\boldsymbol{h}_t^{\rightarrow} = \varphi(\boldsymbol{W}_{xh}^{\rightarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\rightarrow}\boldsymbol{h}_{t-1}^{\rightarrow} + \boldsymbol{b}_h^{\rightarrow}) \tag{23}$$

Bidirectional seq2vec model

Bidirectional models allow the output to depend on the entire sequence (i.e. the hidden state depends on past and future contexts):

$$\begin{aligned}
\boldsymbol{h}_t^{\rightarrow} &= \varphi(\boldsymbol{W}_{xh}^{\rightarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\rightarrow}\boldsymbol{h}_{t-1}^{\rightarrow} + \boldsymbol{b}_h^{\rightarrow}) \\
\boldsymbol{h}_t^{\leftarrow} &= \varphi(\boldsymbol{W}_{xh}^{\leftarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\leftarrow}\boldsymbol{h}_{t+1}^{\leftarrow} + \boldsymbol{b}_h^{\leftarrow})
\end{aligned} \tag{23}$$

Bidirectional seq2vec model

Bidirectional models allow the output to depend on the entire sequence (i.e. the hidden state depends on past and future contexts):

$$\boldsymbol{h}_t^{\rightarrow} = \varphi(\boldsymbol{W}_{xh}^{\rightarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\rightarrow}\boldsymbol{h}_{t-1}^{\rightarrow} + \boldsymbol{b}_h^{\rightarrow}) \tag{23}$$

$$\boldsymbol{h}_t^{\leftarrow} = \varphi(\boldsymbol{W}_{xh}^{\leftarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\leftarrow}\boldsymbol{h}_{t+1}^{\leftarrow} + \boldsymbol{b}_h^{\leftarrow}) \tag{24}$$

Bidirectional seq2vec model

Bidirectional models allow the output to depend on the entire sequence (i.e. the hidden state depends on past and future contexts):

$$\boldsymbol{h}_t^{\rightarrow} = \varphi(\boldsymbol{W}_{xh}^{\rightarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\rightarrow}\boldsymbol{h}_{t-1}^{\rightarrow} + \boldsymbol{b}_h^{\rightarrow}) \tag{23}$$

$$\boldsymbol{h}_t^{\leftarrow} = \varphi(\boldsymbol{W}_{xh}^{\leftarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\leftarrow}\boldsymbol{h}_{t+1}^{\leftarrow} + \boldsymbol{b}_h^{\leftarrow}) \tag{24}$$

- State at time t: $\boldsymbol{h}_t = [\boldsymbol{h}_t^{\rightarrow}, \boldsymbol{h}_t^{\leftarrow}]$

Bidirectional seq2vec model

Bidirectional models allow the output to depend on the entire sequence (i.e. the hidden state depends on past and future contexts):

$$\boldsymbol{h}_t^{\rightarrow} = \varphi(\boldsymbol{W}_{xh}^{\rightarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\rightarrow}\boldsymbol{h}_{t-1}^{\rightarrow} + \boldsymbol{b}_h^{\rightarrow}) \quad (23)$$

$$\boldsymbol{h}_t^{\leftarrow} = \varphi(\boldsymbol{W}_{xh}^{\leftarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\leftarrow}\boldsymbol{h}_{t+1}^{\leftarrow} + \boldsymbol{b}_h^{\leftarrow}) \quad (24)$$

- State at time t: $\boldsymbol{h}_t = [\boldsymbol{h}_t^{\rightarrow}, \boldsymbol{h}_t^{\leftarrow}]$
- Final classification then given by:

Bidirectional seq2vec model

Bidirectional models allow the output to depend on the entire sequence (i.e. the hidden state depends on past and future contexts):

$$\boldsymbol{h}_t^{\rightarrow} = \varphi(\boldsymbol{W}_{xh}^{\rightarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\rightarrow}\boldsymbol{h}_{t-1}^{\rightarrow} + \boldsymbol{b}_h^{\rightarrow}) \tag{23}$$

$$\boldsymbol{h}_t^{\leftarrow} = \varphi(\boldsymbol{W}_{xh}^{\leftarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\leftarrow}\boldsymbol{h}_{t+1}^{\leftarrow} + \boldsymbol{b}_h^{\leftarrow}) \tag{24}$$

- State at time t: $\boldsymbol{h}_t = [\boldsymbol{h}_t^{\rightarrow}, \boldsymbol{h}_t^{\leftarrow}]$
- Final classification then given by:

$$p(y|\boldsymbol{x}_{1:T}) = \mathrm{Cat}(y|\boldsymbol{W}\mathcal{S}(\overline{\boldsymbol{h}})) \tag{25}$$

Bidirectional seq2vec model

Bidirectional models allow the output to depend on the entire sequence (i.e. the hidden state depends on past and future contexts):

$$\boldsymbol{h}_t^{\rightarrow} = \varphi(\boldsymbol{W}_{xh}^{\rightarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\rightarrow}\boldsymbol{h}_{t-1}^{\rightarrow} + \boldsymbol{b}_h^{\rightarrow}) \tag{23}$$

$$\boldsymbol{h}_t^{\leftarrow} = \varphi(\boldsymbol{W}_{xh}^{\leftarrow}\boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\leftarrow}\boldsymbol{h}_{t+1}^{\leftarrow} + \boldsymbol{b}_h^{\leftarrow}) \tag{24}$$

- State at time t: $\boldsymbol{h}_t = [\boldsymbol{h}_t^{\rightarrow}, \boldsymbol{h}_t^{\leftarrow}]$
- Final classification then given by:

$$p(y|\boldsymbol{x}_{1:T}) = \mathrm{Cat}(y|\boldsymbol{W}\mathcal{S}(\overline{\boldsymbol{h}})) \tag{25}$$

where $\overline{\boldsymbol{h}} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{h}_t$

Bidirectional seq2vec model

Bidirectional models allow the output to depend on the entire sequence (i.e. the hidden state depends on past and future contexts):

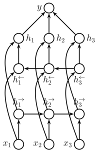
$$\boldsymbol{h}_t^{\rightarrow} = \varphi(\boldsymbol{W}_{xh}^{\rightarrow} \boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\rightarrow} \boldsymbol{h}_{t-1}^{\rightarrow} + \boldsymbol{b}_h^{\rightarrow}) \tag{23}$$

$$\boldsymbol{h}_t^{\leftarrow} = \varphi(\boldsymbol{W}_{xh}^{\leftarrow} \boldsymbol{x}_t + \boldsymbol{W}_{hh}^{\leftarrow} \boldsymbol{h}_{t+1}^{\leftarrow} + \boldsymbol{b}_h^{\leftarrow}) \tag{24}$$

- State at time t: $\boldsymbol{h}_t = [\boldsymbol{h}_t^{\rightarrow}, \boldsymbol{h}_t^{\leftarrow}]$
- Final classification then given by:

$$p(y|\boldsymbol{x}_{1:T}) = \text{Cat}(y|\boldsymbol{W}\mathcal{S}(\overline{\boldsymbol{h}})) \tag{25}$$

where $\overline{\boldsymbol{h}} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{h}_t$



Long-term dependencies in RNNs

Long-term dependencies in RNNs

- RNNs involve multiple compositions of the same function, e.g. in a simple network:

Long-term dependencies in RNNs

- RNNs involve multiple compositions of the same function, e.g. in a simple network:

$$\boldsymbol{h}_t = \boldsymbol{W}^T \boldsymbol{h}_{t-1} \tag{26}$$

Long-term dependencies in RNNs

- RNNs involve multiple compositions of the same function, e.g. in a simple network:

$$\boldsymbol{h}_t = \boldsymbol{W}^T \boldsymbol{h}_{t-1} \tag{26}$$

- Given the above recurrence relation, we can then write:

Long-term dependencies in RNNs

- RNNs involve multiple compositions of the same function, e.g. in a simple network:

$$\boldsymbol{h}_t = \boldsymbol{W}^T \boldsymbol{h}_{t-1} \tag{26}$$

- Given the above recurrence relation, we can then write:

$$\boldsymbol{h}_t = (\boldsymbol{W}^t)^T \boldsymbol{h}^{(0)} \tag{27}$$

Long-term dependencies in RNNs

- RNNs involve multiple compositions of the same function, e.g. in a simple network:

$$\boldsymbol{h}_t = \boldsymbol{W}^T \boldsymbol{h}_{t-1} \tag{26}$$

- Given the above recurrence relation, we can then write:

$$\boldsymbol{h}_t = (\boldsymbol{W}^t)^T \boldsymbol{h}^{(0)} \tag{27}$$

$$\boldsymbol{h}_t = \boldsymbol{Q}^T \Lambda^t \boldsymbol{Q} \boldsymbol{h}^{(0)} \tag{28}$$

Long-term dependencies in RNNs

- RNNs involve multiple compositions of the same function, e.g. in a simple network:

$$\boldsymbol{h}_t = \boldsymbol{W}^T \boldsymbol{h}_{t-1} \tag{26}$$

- Given the above recurrence relation, we can then write:

$$\boldsymbol{h}_t = (\boldsymbol{W}^t)^T \boldsymbol{h}^{(0)} \tag{27}$$

$$\boldsymbol{h}_t = \boldsymbol{Q}^T \boldsymbol{\Lambda}^t \boldsymbol{Q}\boldsymbol{h}^{(0)} \tag{28}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues λ_i

Long-term dependencies in RNNs

- RNNs involve multiple compositions of the same function, e.g. in a simple network:

$$\boldsymbol{h}_t = \boldsymbol{W}^T \boldsymbol{h}_{t-1} \tag{26}$$

- Given the above recurrence relation, we can then write:

$$\boldsymbol{h}_t = (\boldsymbol{W}^t)^T \boldsymbol{h}^{(0)} \tag{27}$$

$$\boldsymbol{h}_t = \boldsymbol{Q}^T \boldsymbol{\Lambda}^t \boldsymbol{Q}\boldsymbol{h}^{(0)} \tag{28}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues λ_i

- Thus the eigenvalues are raised to the power of t

Long-term dependencies in RNNs

- RNNs involve multiple compositions of the same function, e.g. in a simple network:

$$\boldsymbol{h}_t = \boldsymbol{W}^T \boldsymbol{h}_{t-1} \tag{26}$$

- Given the above recurrence relation, we can then write:

$$\begin{aligned} \boldsymbol{h}_t &= (\boldsymbol{W}^t)^T \boldsymbol{h}^{(0)} & (27) \\ \boldsymbol{h}_t &= \boldsymbol{Q}^T \boldsymbol{\Lambda}^t \boldsymbol{Q} \boldsymbol{h}^{(0)} & (28) \end{aligned}$$

  where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues λ_i
- Thus the eigenvalues are raised to the power of t
  - $\lambda_i < 1$: decay to zero (vanishing gradients)
  - $\lambda_i > 1$: exploding gradients

RNN considerations

RNN considerations

- RNNs are fitted via the **backpropagation through time** (BPTT) algorithm
  - computationally expensive

RNN considerations

- RNNs are fitted via the **backpropagation through time** (BPTT) algorithm
  - computationally expensive
- Challenging to learn **long-term dependencies** due to vanishing/exploding gradients.

RNN considerations

- RNNs are fitted via the **backpropagation through time** (BPTT) algorithm
  - computationally expensive
- Challenging to learn **long-term dependencies** due to vanishing/exploding gradients. Strategies:

RNN considerations

- RNNs are fitted via the **backpropagation through time** (BPTT) algorithm
  - computationally expensive
- Challenging to learn **long-term dependencies** due to vanishing/exploding gradients. Strategies:
  - skip connections across time

RNN considerations

- RNNs are fitted via the **backpropagation through time** (BPTT) algorithm
  - computationally expensive
- Challenging to learn **long-term dependencies** due to vanishing/exploding gradients. Strategies:
  - skip connections across time
  - leaky units across different time scales (via linear self-connections that weight information from the past)

RNN considerations

- RNNs are fitted via the **backpropagation through time** (BPTT) algorithm
  - computationally expensive
- Challenging to learn **long-term dependencies** due to vanishing/exploding gradients. Strategies:
  - skip connections across time
  - leaky units across different time scales (via linear self-connections that weight information from the past)

$$\boldsymbol{h}_t \leftarrow \alpha \boldsymbol{h}_{t-1} + (1-\alpha)\boldsymbol{h}_t \tag{29}$$

RNN considerations

- RNNs are fitted via the **backpropagation through time** (BPTT) algorithm
  - computationally expensive
- Challenging to learn **long-term dependencies** due to vanishing/exploding gradients. Strategies:
  - skip connections across time
  - leaky units across different time scales (via linear self-connections that weight information from the past)

$$\boldsymbol{h}_t \leftarrow \alpha \boldsymbol{h}_{t-1} + (1-\alpha)\boldsymbol{h}_t \tag{29}$$

    where α is the weight
  - gradient clipping

RNN considerations

- RNNs are fitted via the **backpropagation through time** (BPTT) algorithm
  - computationally expensive
- Challenging to learn **long-term dependencies** due to vanishing/exploding gradients. Strategies:
  - skip connections across time
  - leaky units across different time scales (via linear self-connections that weight information from the past)

$$\boldsymbol{h}_t \leftarrow \alpha \boldsymbol{h}_{t-1} + (1-\alpha)\boldsymbol{h}_t \tag{29}$$

    where α is the weight
  - gradient clipping
  - train RNN to reset irrelevant states to zero at various points in sequence (via gated units)

Gated RNNs

Gated RNNs

Gated RNNs are a generalization of leaky units that allow for time-dependent variation of self-connection weights.

Gated RNNs

Gated RNNs are a generalization of leaky units that allow for time-dependent variation of self-connection weights.

- In leaky units, the weights are either manually set or learned as parameters, in order to accumlate information
- Gated RNNs enable the “forgetting” of old states
- Most effective gated RNNs in use:

Gated RNNs

Gated RNNs are a generalization of leaky units that allow for time-dependent variation of self-connection weights.

- In leaky units, the weights are either manually set or learned as parameters, in order to accumlate information
- Gated RNNs enable the "forgetting" of old states
- Most effective gated RNNs in use:
  - long short-term memory (LSTM); Hochreiter and Schmidhuber, 1997

Gated RNNs

Gated RNNs are a generalization of leaky units that allow for time-dependent variation of self-connection weights.

- In leaky units, the weights are either manually set or learned as parameters, in order to accumlate information
- Gated RNNs enable the "forgetting" of old states
- Most effective gated RNNs in use:
  - long short-term memory (LSTM); Hochreiter and Schmidhuber, 1997
  - gated recurrent unit (GRU) Cho et al., 2014

Gated RNNs

Gated RNNs are a generalization of leaky units that allow for time-dependent variation of self-connection weights.

- In leaky units, the weights are either manually set or learned as parameters, in order to accumlate information
- Gated RNNs enable the "forgetting" of old states
- Most effective gated RNNs in use:
  - long short-term memory (LSTM); Hochreiter and Schmidhuber, 1997
  - gated recurrent unit (GRU) Cho et al., 2014
  - The LSTM cell has four neural network layers (compared to one layer in the standard RNN)

Gated RNNs

Gated RNNs are a generalization of leaky units that allow for time-dependent variation of self-connection weights.

- In leaky units, the weights are either manually set or learned as parameters, in order to accumlate information
- Gated RNNs enable the "forgetting" of old states
- Most effective gated RNNs in use:
  - long short-term memory (LSTM); Hochreiter and Schmidhuber, 1997
  - gated recurrent unit (GRU) Cho et al., 2014
  - The LSTM cell has four neural network layers (compared to one layer in the standard RNN)

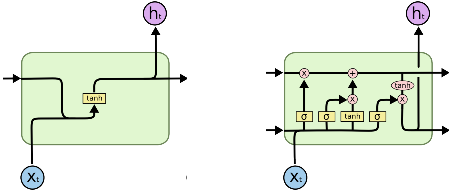


Figure: L: hidden unit in standard RNN; R: hidden unit in LSTM.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

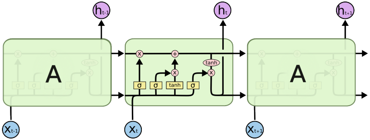
LSTM chains and blocks

LSTM chains and blocks

- There are as many LSTM cells are there are hidden units in current implementations

LSTM chains and blocks

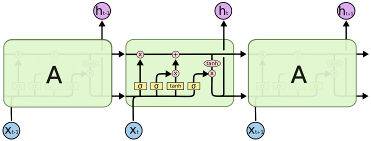
- There are as many LSTM cells are there are hidden units in current implementations



△ Repeating module in LSTM network (https://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM chains and blocks

- There are as many LSTM cells are there are hidden units in current implementations

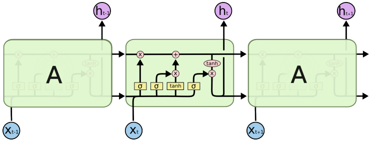


△ Repeating module in LSTM network (https://colah.github.io/posts/2015-08-Understanding-LSTMs/)

- A chain of LSTM cells in a network may be referred to as "layer" or "block" (usage/terminology differs)

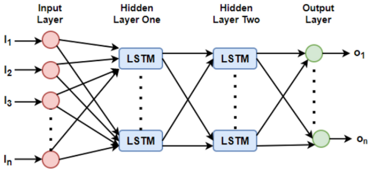
LSTM chains and blocks

- There are as many LSTM cells are there are hidden units in current implementations



△ Repeating module in LSTM network (https://colah.github.io/posts/2015-08-Understanding-LSTMs/)

- A chain of LSTM cells in a network may be referred to as "layer" or "block" (usage/terminology differs)



△ LSTM network with 2 blocks of LSTM cells

(https://link.springer.com/article/10.1007/s42452-021-04421-x)

LSTM gates

LSTM gates

Gates allow the LSTM to decide which signal to pass or block by outputting a number in the interval $[0, 1]$ (via a sigmoid activation)

LSTM gates

Gates allow the LSTM to decide which signal to pass or block by outputting a number in the interval $[0, 1]$ (via a sigmoid activation)

The LSTM cell consists of three gates:

- **Forget gate:** Decides what information will be discarded from cell state. Contained in sigmoid layer that outputs number between 0 and 1 for each number in cell state $\boldsymbol{c}_{t-1}$

LSTM gates

Gates allow the LSTM to decide which signal to pass or block by outputting a number in the interval $[0, 1]$ (via a sigmoid activation)

The LSTM cell consists of three gates:

- **Forget gate:** Decides what information will be discarded from cell state. Contained in sigmoid layer that outputs number between 0 and 1 for each number in cell state $\boldsymbol{c}_{t-1}$
- **Input gate:** Decides what new information to store in cell state. Comprises
  a. sigmoid layer which decides values to update

LSTM gates

Gates allow the LSTM to decide which signal to pass or block by outputting a number in the interval $[0, 1]$ (via a sigmoid activation)

The LSTM cell consists of three gates:

- **Forget gate:** Decides what information will be discarded from cell state. Contained in sigmoid layer that outputs number between 0 and 1 for each number in cell state $\boldsymbol{c}_{t-1}$
- **Input gate:** Decides what new information to store in cell state. Comprises
  - a. sigmoid layer which decides values to update
  - b. tanh layer to create new candidate values for the state

LSTM gates

Gates allow the LSTM to decide which signal to pass or block by outputting a number in the interval $[0, 1]$ (via a sigmoid activation)

The LSTM cell consists of three gates:

- **Forget gate:** Decides what information will be discarded from cell state. Contained in sigmoid layer that outputs number between 0 and 1 for each number in cell state $\boldsymbol{c}_{t-1}$
- **Input gate:** Decides what new information to store in cell state. Comprises
  - a. sigmoid layer which decides values to update
  - b. tanh layer to create new candidate values for the state
- **Output gate:** Decides what information is passed from cell state to output. Comprises
  - a. sigmoid layer to decide portion of cell state to retain

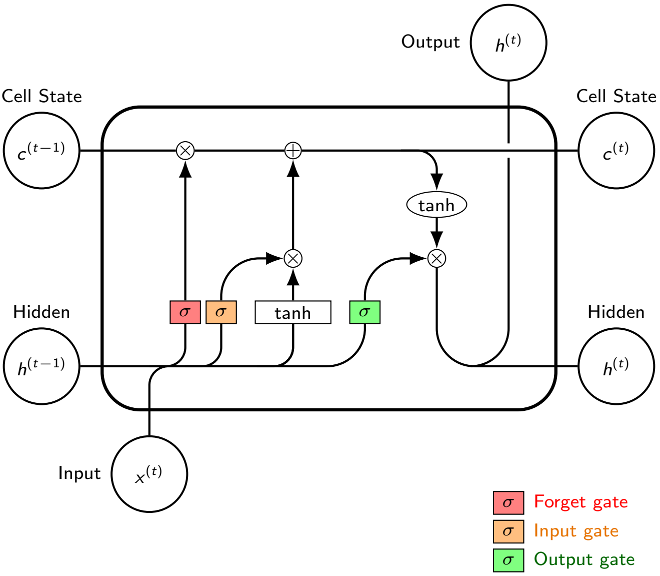
LSTM gates

Gates allow the LSTM to decide which signal to pass or block by outputting a number in the interval $[0, 1]$ (via a sigmoid activation)

The LSTM cell consists of three gates:

- **Forget gate:** Decides what information will be discarded from cell state. Contained in sigmoid layer that outputs number between 0 and 1 for each number in cell state $\boldsymbol{c}_{t-1}$
- **Input gate:** Decides what new information to store in cell state. Comprises
  - a sigmoid layer which decides values to update
  - b tanh layer to create new candidate values for the state
- **Output gate:** Decides what information is passed from cell state to output. Comprises
  - a sigmoid layer to decide portion of cell state to retain
  - b tanh later which admits state values in $[-1, 1]$

LSTM cell anatomy



Forget gate

Forget gate

The forget gate determines what information should be discarded from the cell.

Forget gate

The forget gate determines what information should be discarded from the cell. The gate unit is given by:

$$f_{t,i} = \boldsymbol{\sigma}\left(b_i^f + \sum_j U_{i,j}^f x_{t,j} + \sum_j W_{i,k}^f h_{t-1,k}\right) \tag{30}$$

Forget gate

The forget gate determines what information should be discarded from the cell. The gate unit is given by:

$$f_{t,i} = \boldsymbol{\sigma}\left(b_i^f + \sum_j U_{i,j}^f x_{t,j} + \sum_j W_{i,k}^f h_{t-1,k}\right) \tag{30}$$

where:

- $f_{t,i}$: forget gate value for timestep t and cell i (between 0 and 1)
- $\boldsymbol{x}_t$: current input vector

Forget gate

The forget gate determines what information should be discarded from the cell. The gate unit is given by:

$$f_{t,i} = \boldsymbol{\sigma} \left(b_i^f + \sum_j U_{i,j}^f x_{t,j} + \sum_j W_{i,k}^f h_{t-1,k} \right) \tag{30}$$

where:

- $f_{t,i}$: forget gate value for timestep t and cell i (between 0 and 1)
- $\boldsymbol{x}_t$: current input vector
- $\boldsymbol{h}_{t-1}$: hidden state from previous cell

Forget gate

The forget gate determines what information should be discarded from the cell. The gate unit is given by:

$$f_{t,i} = \boldsymbol{\sigma}\left(b_i^f + \sum_j U_{i,j}^f x_{t,j} + \sum_j W_{i,k}^f h_{t-1,k}\right) \tag{30}$$

where:

- $f_{t,i}$: forget gate value for timestep t and cell i (between 0 and 1)
- $\boldsymbol{x}_t$: current input vector
- $\boldsymbol{h}_{t-1}$: hidden state from previous cell
- $\boldsymbol{b}^f$, $\boldsymbol{U}^f$, $\boldsymbol{W}^f$: biases, input weights and recurrent weights for the forget gates

Input gate

Input gate

The input gate determines what new information to include in the cell.

Input gate

The input gate determines what new information to include in the cell. Its unit is given by:

Input gate

The input gate determines what new information to include in the cell. Its unit is given by:

$$g_{t,i} = \boldsymbol{\sigma}\left(b_i^g + \sum_j U_{i,j}^g x_{t,j} + \sum_j W_{i,k}^g h_{t-1,k}\right) \tag{31}$$

Input gate

The input gate determines what new information to include in the cell. Its unit is given by:

$$g_{t,i} = \boldsymbol{\sigma}\left(b_i^g + \sum_j U_{i,j}^g x_{t,j} + \sum_j W_{i,k}^g h_{t-1,k}\right) \tag{31}$$

where:

- $\boldsymbol{b}^g$, $\boldsymbol{U}^g$, $\boldsymbol{W}^g$: biases, input weights and recurrent weights for the input gates
- $\boldsymbol{h}_{t-1}$: hidden state from previous cell

Input gate

The input gate determines what new information to include in the cell. Its unit is given by:

$$g_{t,i} = \boldsymbol{\sigma} \left(b_i^g + \sum_j U_{i,j}^g x_{t,j} + \sum_j W_{i,k}^g h_{t-1,k} \right) \tag{31}$$

where:

- $\boldsymbol{b}^g$, $\boldsymbol{U}^g$, $\boldsymbol{W}^g$: biases, input weights and recurrent weights for the input gates
- $\boldsymbol{h}_{t-1}$: hidden state from previous cell

State update

State update

The cell state update is given by:

$$c_{t,i} = f_{t,i} c_{t-1,i} + g_{t,i} \tanh\left(b_i + \sum_j U_{i,j} x_{t,j} + \sum_j W_{i,k} h_{t-1,j} \right) \tag{32}$$

State update

The cell state update is given by:

$$c_{t,i} = f_{t,i} c_{t-1,i} + g_{t,i} \tanh\left(b_i + \sum_j U_{i,j} x_{t,j} + \sum_j W_{i,k} h_{t-1,j} \right) \tag{32}$$

- $\boldsymbol{b}$, $\boldsymbol{U}$, $\boldsymbol{W}$ are biases, input weights and recurrent weights, respectively, into the LSTM cell
- $c_{t-1,i}$ is the cell state for the prior timestep
- The term in purple represents the signal from new information $\boldsymbol{x}_t$ that may be included in the cell state update. Let us call it $\tilde{\boldsymbol{c}}_t$
- Thus, we may write the cell state update as:

State update

The cell state update is given by:

$$c_{t,i} = f_{t,i}c_{t-1,i} + g_{t,i}\tanh\left(b_i + \sum_j U_{i,j}x_{t,j} + \sum_j W_{i,k}h_{t-1,j}\right) \tag{32}$$

- $\boldsymbol{b}$, $\boldsymbol{U}$, $\boldsymbol{W}$ are biases, input weights and recurrent weights, respectively, into the LSTM cell
- $c_{t-1,i}$ is the cell state for the prior timestep
- The term in purple represents the signal from new information $\boldsymbol{x}_t$ that may be included in the cell state update. Let us call it $\tilde{\boldsymbol{c}}_t$
- Thus, we may write the cell state update as:

$$c_{t,i} = f_{t,i}c_{t-1,i} + g_{t,i}\tilde{c}_{t,i} \tag{33}$$

- In this form, we explicitly how the cell state is composed of a weighted sum of the prior cell state (accumulated information) and new inputs

Output gate

Output gate

The output gate unit is given by:

$$q_{t,i} = \boldsymbol{\sigma}\left(b_i^o + \sum_j U_{i,j}^o x_{t,j} + \sum_j W_{i,j}^o h_{t-1,j}\right) \tag{34}$$

Output gate

The output gate unit is given by:

$$q_{t,i} = \boldsymbol{\sigma}\left(b_i^o + \sum_j U_{i,j}^o x_{t,j} + \sum_j W_{i,j}^o h_{t-1,j}\right) \tag{34}$$

And the final output (hidden state) of the LSTM cell is:

$$h_{t,i} = \tanh(c_{t,i}) q_{t,i} \tag{35}$$

LSTM: putting it all together

LSTM: putting it all together

The complete set of LSTM update equations is given by:

LSTM: putting it all together

The complete set of LSTM update equations is given by:

$$\boldsymbol{F}_t = \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f) \tag{36}$$

LSTM: putting it all together

The complete set of LSTM update equations is given by:

$$\boldsymbol{F}_t = \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f) \quad (36)$$

$$\boldsymbol{I}_t = \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i) \quad (37)$$

LSTM: putting it all together

The complete set of LSTM update equations is given by:

$$\boldsymbol{F}_t = \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f) \tag{36}$$

$$\boldsymbol{I}_t = \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i) \tag{37}$$

$$\boldsymbol{O}_t = \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o) \tag{38}$$

LSTM: putting it all together

The complete set of LSTM update equations is given by:

$$\begin{aligned}
\boldsymbol{F}_t &= \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f) && (36) \\
\boldsymbol{I}_t &= \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i) && (37) \\
\boldsymbol{O}_t &= \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o) && (38) \\
\tilde{\boldsymbol{C}}_t &= \tanh(\boldsymbol{X}_t \boldsymbol{W}_{xc} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hc} + \boldsymbol{b}_c) && (39)
\end{aligned}$$

LSTM: putting it all together

The complete set of LSTM update equations is given by:

$$\boldsymbol{F}_t = \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f) \tag{36}$$

$$\boldsymbol{I}_t = \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i) \tag{37}$$

$$\boldsymbol{O}_t = \boldsymbol{\sigma}(\boldsymbol{X}_t \boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o) \tag{38}$$

$$\tilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{X}_t \boldsymbol{W}_{xc} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hc} + \boldsymbol{b}_c) \tag{39}$$

$$\boldsymbol{C}_t = \boldsymbol{F}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \odot \tilde{\boldsymbol{C}}_t \tag{40}$$

LSTM: putting it all together

The complete set of LSTM update equations is given by:

$$\boldsymbol{F}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f) \tag{36}$$

$$\boldsymbol{I}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i) \tag{37}$$

$$\boldsymbol{O}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o) \tag{38}$$

$$\tilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{X}_t \boldsymbol{W}_{xc} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hc} + \boldsymbol{b}_c) \tag{39}$$

$$\boldsymbol{C}_t = \boldsymbol{F}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \odot \tilde{\boldsymbol{C}}_t \tag{40}$$

$$\boldsymbol{H}_t = \boldsymbol{O}_t \odot \tanh(\boldsymbol{C}_t) \tag{41}$$

LSTM: putting it all together

The complete set of LSTM update equations is given by:

$$\boldsymbol{F}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f) \tag{36}$$

$$\boldsymbol{I}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i) \tag{37}$$

$$\boldsymbol{O}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o) \tag{38}$$

$$\tilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{X}_t \boldsymbol{W}_{xc} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hc} + \boldsymbol{b}_c) \tag{39}$$

$$\boldsymbol{C}_t = \boldsymbol{F}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \odot \tilde{\boldsymbol{C}}_t \tag{40}$$

$$\boldsymbol{H}_t = \boldsymbol{O}_t \odot \tanh(\boldsymbol{C}_t) \tag{41}$$

where:

- $\boldsymbol{F}_t$, $\boldsymbol{I}_t$, $\boldsymbol{O}_t$: forget, input and output gate vectors at time t
- $\boldsymbol{C}_t$: cell state vector at time t
- $\boldsymbol{H}_t$: hidden state (output) vector at time t
- $\odot$: element-wise (Hadamard) product

Attention

Attention

- Typical neural networks process all parts of the input with equal importance:

Attention

- Typical neural networks process all parts of the input with equal importance:

$$\boldsymbol{z} = \varphi(\boldsymbol{W}\boldsymbol{v}), \quad \boldsymbol{v} \in \mathbb{R}^{v}, \boldsymbol{W} \in \mathbb{R}^{v' \times v} \tag{42}$$

Attention

- Typical neural networks process all parts of the input with equal importance:

$$\boldsymbol{z} = \varphi(\boldsymbol{W}\boldsymbol{v}), \quad \boldsymbol{v} \in \mathbb{R}^{v}, \boldsymbol{W} \in \mathbb{R}^{v' \times v} \tag{42}$$

- Attention mechanisms were introduced to allow flexibility: i.e. for models to dynamically focus on the most relevant portion of the input when generating each part of the output.

Attention

- Typical neural networks process all parts of the input with equal importance:

$$\boldsymbol{z} = \varphi(\boldsymbol{W}\boldsymbol{v}), \quad \boldsymbol{v} \in \mathbb{R}^{v}, \boldsymbol{W} \in \mathbb{R}^{v' \times v} \tag{42}$$

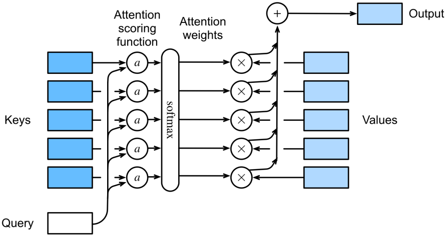
- Attention mechanisms were introduced to allow flexibility: i.e. for models to dynamically focus on the most relevant portion of the input when generating each part of the output.
- This is typically done by computing a weighted sum of the input features $\boldsymbol{v}_i$, where the weights are learned based on the input itself.

Attention

- Typical neural networks process all parts of the input with equal importance:

$$\boldsymbol{z} = \varphi(\boldsymbol{W}\boldsymbol{v}), \quad \boldsymbol{v} \in \mathbb{R}^{v}, \boldsymbol{W} \in \mathbb{R}^{v' \times v} \tag{42}$$

- Attention mechanisms were introduced to allow flexibility: i.e. for models to dynamically focus on the most relevant portion of the input when generating each part of the output.
- This is typically done by computing a weighted sum of the input features $\boldsymbol{v}_i$, where the weights are learned based on the input itself.



△ Attention as weighted sum of values (Source: https://github.com/probml/pml-book/blob/main/book1-figures/Figure_15.16.pdf)

Attention scores

Attention scores

- Given m values $\boldsymbol{V} \in \mathbb{R}^{m \times v}$

Attention scores

- Given m values $\boldsymbol{V} \in \mathbb{R}^{m \times v}$
- input query vector $\boldsymbol{q} \in \mathbb{R}^{q}$

Attention scores

- Given m values $\boldsymbol{V} \in \mathbb{R}^{m \times v}$
- input query vector $\boldsymbol{q} \in \mathbb{R}^{q}$
- m keys $\boldsymbol{K} \in \mathbb{R}^{m \times k}$

Attention scores

- Given m values $\boldsymbol{V} \in \mathbb{R}^{m \times v}$
- input query vector $\boldsymbol{q} \in \mathbb{R}^{q}$
- m keys $\boldsymbol{K} \in \mathbb{R}^{m \times k}$
- Attention output is given by:

Attention scores

- Given m values $\boldsymbol{V} \in \mathbb{R}^{m \times v}$
- input query vector $\boldsymbol{q} \in \mathbb{R}^{q}$
- m keys $\boldsymbol{K} \in \mathbb{R}^{m \times k}$
- Attention output is given by:

$$\text{Attention}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{m} \alpha_i \boldsymbol{v}_i \tag{43}$$

Attention scores

- Given m values $\boldsymbol{V} \in \mathbb{R}^{m \times v}$
- input query vector $\boldsymbol{q} \in \mathbb{R}^{q}$
- m keys $\boldsymbol{K} \in \mathbb{R}^{m \times k}$
- Attention output is given by:

$$\text{Attention}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{m} \alpha_i \boldsymbol{v}_i \tag{43}$$

where the attention weights α_i are computed as:

Attention scores

- Given m values $\boldsymbol{V} \in \mathbb{R}^{m \times v}$
- input query vector $\boldsymbol{q} \in \mathbb{R}^{q}$
- m keys $\boldsymbol{K} \in \mathbb{R}^{m \times k}$
- Attention output is given by:

$$\text{Attention}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{m} \alpha_i \boldsymbol{v}_i \tag{43}$$

where the attention weights α_i are computed as:

$$\alpha_i = \frac{\exp(a(\boldsymbol{q}, \boldsymbol{k}_i))}{\sum_{j=1}^{m} \exp(a(\boldsymbol{q}, \boldsymbol{k}_j))} \tag{44}$$

Attention scores

- Given m values $\boldsymbol{V} \in \mathbb{R}^{m \times v}$
- input query vector $\boldsymbol{q} \in \mathbb{R}^{q}$
- m keys $\boldsymbol{K} \in \mathbb{R}^{m \times k}$
- Attention output is given by:

$$\text{Attention}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{m} \alpha_i \boldsymbol{v}_i \tag{43}$$

where the attention weights α_i are computed as:

$$\alpha_i = \frac{\exp(a(\boldsymbol{q}, \boldsymbol{k}_i))}{\sum_{j=1}^{m} \exp(a(\boldsymbol{q}, \boldsymbol{k}_j))} \tag{44}$$

and $a(\boldsymbol{q}, \boldsymbol{k}_i)$ is a score function that measures the similarity between the query $\boldsymbol{q}$ and key vectors $\boldsymbol{k}_i$.

Attention scores

- Given m values $\boldsymbol{V} \in \mathbb{R}^{m \times v}$
- input query vector $\boldsymbol{q} \in \mathbb{R}^{q}$
- m keys $\boldsymbol{K} \in \mathbb{R}^{m \times k}$
- Attention output is given by:

$$\text{Attention}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{m} \alpha_i \boldsymbol{v}_i \tag{43}$$

where the attention weights α_i are computed as:

$$\alpha_i = \frac{\exp(a(\boldsymbol{q}, \boldsymbol{k}_i))}{\sum_{j=1}^{m} \exp(a(\boldsymbol{q}, \boldsymbol{k}_j))} \tag{44}$$

and $a(\boldsymbol{q}, \boldsymbol{k}_i)$ is a score function that measures the similarity between the query $\boldsymbol{q}$ and key vectors $\boldsymbol{k}_i$.

Commonly used score functions

Commonly used score functions

Common choices for the score function $a(\boldsymbol{q}, \boldsymbol{k}_i)$ include:

Commonly used score functions

Common choices for the score function $a(\boldsymbol{q}, \boldsymbol{k}_i)$ include:

- Dot product:

Commonly used score functions

Common choices for the score function $a(\boldsymbol{q}, \boldsymbol{k}_i)$ include:

- Dot product:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^T \boldsymbol{k}_i \tag{45}$$

Commonly used score functions

Common choices for the score function $a(\boldsymbol{q}, \boldsymbol{k}_i)$ include:

- Dot product:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^T \boldsymbol{k}_i \tag{45}$$

  Often it is scaled by $\sqrt{d}$ to ensure the variance of the score remains 1:

Commonly used score functions

Common choices for the score function $a(\boldsymbol{q}, \boldsymbol{k}_i)$ include:

- Dot product:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^T \boldsymbol{k}_i \tag{45}$$

  Often it is scaled by $\sqrt{d}$ to ensure the variance of the score remains 1:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \frac{\boldsymbol{q}^T \boldsymbol{k}_i}{\sqrt{d}} \tag{46}$$

Commonly used score functions

Common choices for the score function $a(\boldsymbol{q}, \boldsymbol{k}_i)$ include:

- Dot product:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^T \boldsymbol{k}_i \tag{45}$$

  Often it is scaled by $\sqrt{d}$ to ensure the variance of the score remains 1:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \frac{\boldsymbol{q}^T \boldsymbol{k}_i}{\sqrt{d}} \tag{46}$$

- Additive (Bahdanau) attention:

Commonly used score functions

Common choices for the score function $a(\boldsymbol{q}, \boldsymbol{k}_i)$ include:

- Dot product:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^T \boldsymbol{k}_i \tag{45}$$

  Often it is scaled by $\sqrt{d}$ to ensure the variance of the score remains 1:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \frac{\boldsymbol{q}^T \boldsymbol{k}_i}{\sqrt{d}} \tag{46}$$

- Additive (Bahdanau) attention:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{w}_a^T \tanh(\boldsymbol{W}_q \boldsymbol{q} + \boldsymbol{W}_k \boldsymbol{k}_i + \boldsymbol{b}_a) \tag{47}$$

Commonly used score functions

Common choices for the score function $a(\boldsymbol{q}, \boldsymbol{k}_i)$ include:

- Dot product:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^T \boldsymbol{k}_i \tag{45}$$

  Often it is scaled by $\sqrt{d}$ to ensure the variance of the score remains 1:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \frac{\boldsymbol{q}^T \boldsymbol{k}_i}{\sqrt{d}} \tag{46}$$

- Additive (Bahdanau) attention:

$$a(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{w}_a^T \tanh(\boldsymbol{W}_q \boldsymbol{q} + \boldsymbol{W}_k \boldsymbol{k}_i + \boldsymbol{b}_a) \tag{47}$$

  where $\boldsymbol{w}_a$, $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, and $\boldsymbol{b}_a$ are learnable parameters.

Transformers

Transformers

A transformer is a seq2seq model architecture that uses self-attention for the encoder and decoder in place of an RNN.

Transformers

A transformer is a seq2seq model architecture that uses self-attention for the encoder and decoder in place of an RNN.

- Self-attention allows the model to weigh the importance of different words in the input sequence when encoding each word.

Transformers

A transformer is a seq2seq model architecture that uses self-attention for the encoder and decoder in place of an RNN.

- Self-attention allows the model to weigh the importance of different words in the input sequence when encoding each word.

$$\boldsymbol{y}_i = \text{Attention}(\boldsymbol{x}_i, (\boldsymbol{x}_1, \boldsymbol{x}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{x}_n)) \tag{48}$$

Transformers

A transformer is a seq2seq model architecture that uses self-attention for the encoder and decoder in place of an RNN.

- Self-attention allows the model to weigh the importance of different words in the input sequence when encoding each word.

$$\boldsymbol{y}_i = \text{Attention}(\boldsymbol{x}_i, (\boldsymbol{x}_1, \boldsymbol{x}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{x}_n)) \tag{48}$$

where query is $\boldsymbol{x}_i$, and keys and values are all input vectors.

Transformers

A transformer is a seq2seq model architecture that uses self-attention for the encoder and decoder in place of an RNN.

- Self-attention allows the model to weigh the importance of different words in the input sequence when encoding each word.

$$\boldsymbol{y}_i = \text{Attention}(\boldsymbol{x}_i, (\boldsymbol{x}_1, \boldsymbol{x}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{x}_n)) \tag{48}$$

  where query is $\boldsymbol{x}_i$, and keys and values are all input vectors.
- Transformers have been shown to outperform RNNs in various NLP tasks, including machine translation and text generation.

Transformers

A transformer is a seq2seq model architecture that uses self-attention for the encoder and decoder in place of an RNN.

- Self-attention allows the model to weigh the importance of different words in the input sequence when encoding each word.

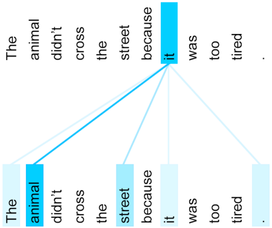
$$\boldsymbol{y}_i = \text{Attention}(\boldsymbol{x}_i, (\boldsymbol{x}_1, \boldsymbol{x}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{x}_n)) \tag{48}$$

  where query is $\boldsymbol{x}_i$, and keys and values are all input vectors.
- Transformers have been shown to outperform RNNs in various NLP tasks, including machine translation and text generation.
- Popular transformer models include BERT, GPT, and T5.

Self-attention for context representation

Self-attention for context representation

Self-attention can allow for improved representation of context.



△ Self-attention for context representation (Source: `https://github.com/probml/pml-book/blob/main/book1-figures/Figure_15.23.png`)

Language models

Language models

Language models are generative sequences of the form:

Language models

Language models are generative sequences of the form:

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | \boldsymbol{x}_{1:t-1}) \tag{49}$$

Language models

Language models are generative sequences of the form:

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | \boldsymbol{x}_{1:t-1}) \tag{49}$$

where x_t is the t-th word in a sequence of T words.

Language models

Language models are generative sequences of the form:

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | \boldsymbol{x}_{1:t-1}) \tag{49}$$

where x_t is the t-th word in a sequence of T words.
Examples include:

- Embeddings from Language Model (ELMo)

Language models

Language models are generative sequences of the form:

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | \boldsymbol{x}_{1:t-1}) \tag{49}$$

where x_t is the t-th word in a sequence of T words.
Examples include:

- Embeddings from Language Model (ELMo)
- Bidirectional Encoder Representations from Transformers (BERT)

Language models

Language models are generative sequences of the form:

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | \boldsymbol{x}_{1:t-1}) \tag{49}$$

where x_t is the t-th word in a sequence of T words.
Examples include:

- Embeddings from Language Model (ELMo)
- Bidirectional Encoder Representations from Transformers (BERT)
- Generative Pre-trained Transformer (GPT) models

Language models

Language models are generative sequences of the form:

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | \boldsymbol{x}_{1:t-1}) \tag{49}$$

where x_t is the t-th word in a sequence of T words.
Examples include:

- Embeddings from Language Model (ELMo)
- Bidirectional Encoder Representations from Transformers (BERT)
- Generative Pre-trained Transformer (GPT) models
- Text-to-text Transfer Transformer (T5) models

Summary

- Recurrent neural networks are designed to learn from sequential (temporal/spatial) data
- The recurrence structure in RNNs renders them susceptible to the vanishing/exploding gradient problem
  - It also makes it challenging for the standard RNN to learn long-term dependencies
- Several approaches have been proposed to address these issues, including the use of gated RNNs
- LSTMs in particular are able to learn when and how much of prior information to include or forget in generating the output at each timestep
  - This is done via the use of gates to control the flow of information
- LSTMs have been successfully applied to handwriting recognition/generation, speech recognition, machine translation, image captioning, among others.

Reading

- Text:**PMLI** 15, DL10
- Note that in DL10, the following symbology used in describing LSTMs (10.1.1.): (Commonly used counterparts that you may find in other literature are parenthesized.)
  - cell state: $\boldsymbol{s}_t$ (alternative: $\boldsymbol{c}_t$)
  - input gate: $\boldsymbol{g}_t$ (alternative: $\boldsymbol{i}_t$)
  - output gate: $\boldsymbol{q}_t$ (alternative: $\boldsymbol{o}_t$)
- I think the $f/c/i/o$ notation is easier to follow than the $f/s/g/q$ used in the DL text
  - However, given that DL uses i as the index for each cell, it is probably less confusing to have i representing two different things.
- An excellent resource for further explanations on how LSTMs work is available on Chris Olah's blog: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`