Sequence Modeling: Recurrent Neural Networks

EE 565: Pattern Recognition and Machine Learning

Matthew Martinez

November 17, 2016

Overview

Family of Neural Networks for Modeling Sequences

- Speech signals, natural language processing, time series prediction, image captioning
- Sequences are defined as $\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(\tau)}$
- Process sequences of variable length

Types of Recurrent Neural Networks

- bidirectional: allows network to learn relationships forward and backward in time
- Echo State Networks
- Long Short-Term Memory (LSTM)

Computational Graphs

Elements of a computational graph

- *Node*: Indicates a variable, i.e. scalar, vector, matrix, tensor, etc

Computational Graphs

Elements of a computational graph

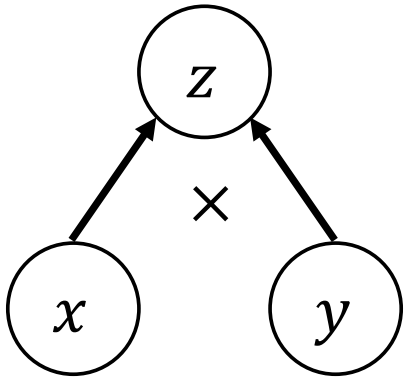
- *Node*: Indicates a variable, i.e. scalar, vector, matrix, tensor, etc
- *Operation*: A simple function of one or more variables
  - Returns only a single output
  - Output can have multiple entries, i.e. vector
  - Functions can be created by composing many operations together.

Computational Graphs

Elements of a computational graph

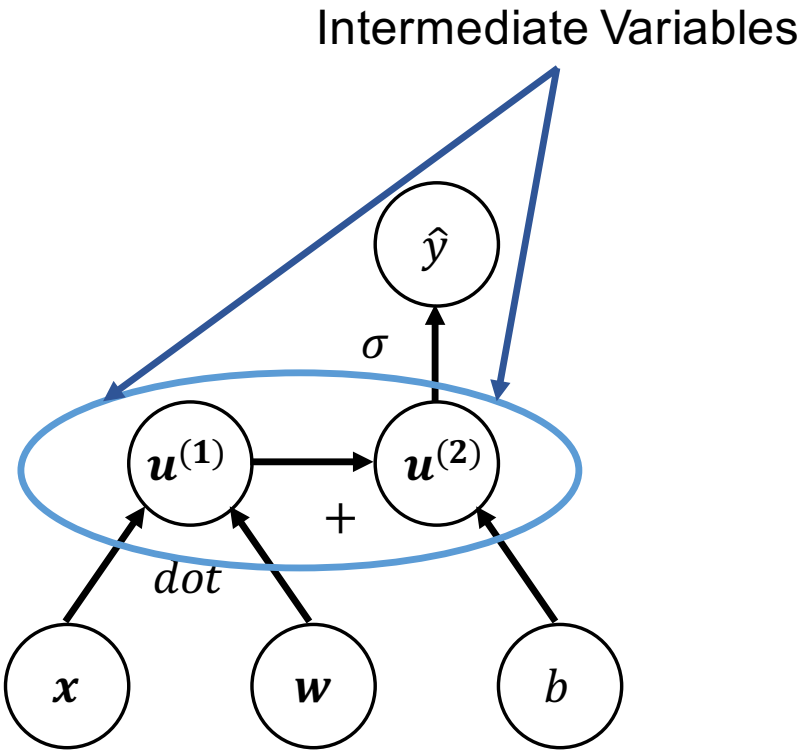
- *Node*: Indicates a variable, i.e. scalar, vector, matrix, tensor, etc
- *Operation*: A simple function of one or more variables
  - Returns only a single output
  - Output can have multiple entries, i.e. vector
  - Functions can be created by composing many operations together.
- Method for quickly computing derivatives
- Computational applications beyond deep learning and machine learning
  - Weather forecasting, numerical stability analysis
- Core abstraction for Theano, TensorFlow, Torch, etc

Example: Multiplication



$$z = xy$$

Example: Logistic Regression



$$\hat{y} = \sigma(\boldsymbol{x}^T\boldsymbol{w} + b)$$

Graph Unfolding

Goal:

- *Unfold* an recurrent or recursive computation into a computation graph
- Computational graph has repetitive structure
- Corresponds to a chain of events
- Results in parameter sharing across a deep network structure

Graph Unfolding

Goal:

- *Unfold* an recurrent or recursive computation into a computation graph
- Computational graph has repetitive structure
- Corresponds to a chain of events
- Results in parameter sharing across a deep network structure

Example: Classical Form of Dynamical System

$$\mathbf{s}^{(t)} = f\left(\mathbf{s}^{(t-1)}; \theta\right) \tag{1}$$

- $\mathbf{s}$:= system state
- $\mathbf{s}$ at time t depends on system state at time $t-1$

Example: Classical Form of Dynamical System

Unfold system state for $\tau = 3$ time steps:

$$\mathbf{s}^{(3)} = f\left(\mathbf{s}^{(2)}; \theta\right) \tag{2}$$

$$= f\left(f\left(\mathbf{s}^{(1)}; \theta\right); \theta\right) \tag{3}$$

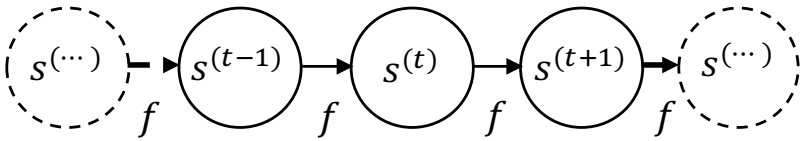
Example: Classical Form of Dynamical System

Unfold system state for $\tau = 3$ time steps:

$$\mathbf{s}^{(3)} = f\left(\mathbf{s}^{(2)}; \theta\right) \tag{2}$$

$$= f\left(f\left(\mathbf{s}^{(1)}; \theta\right); \theta\right) \tag{3}$$

Unfolded computational graph



Example: Dynamic System Driven by External Signal

Example: Dynamic System Driven by External Signal

$$\mathbf{s}^{(t)} = f\left(\mathbf{s}^{(t-1)}, \mathbf{x}^{(t)}; \theta\right) \tag{4}$$

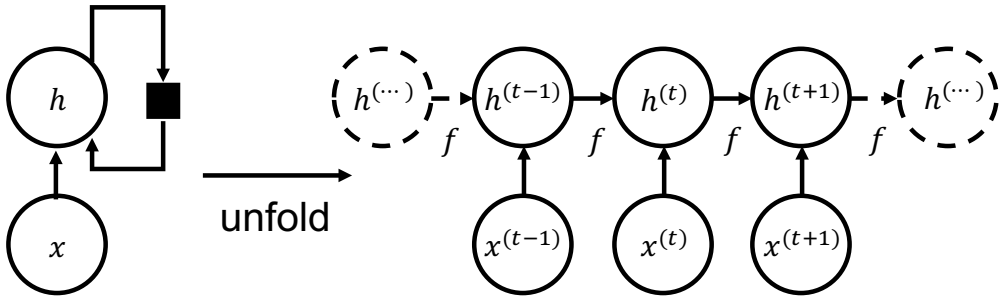
- Any function involving recurrence can be a recurrent neural network
- Use 4 to define hidden units

$$\mathbf{h}^{(t)} = f\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta\right) \tag{5}$$

- Additional architectural features such as output layers can be used to read information out of state $\mathbf{h}$

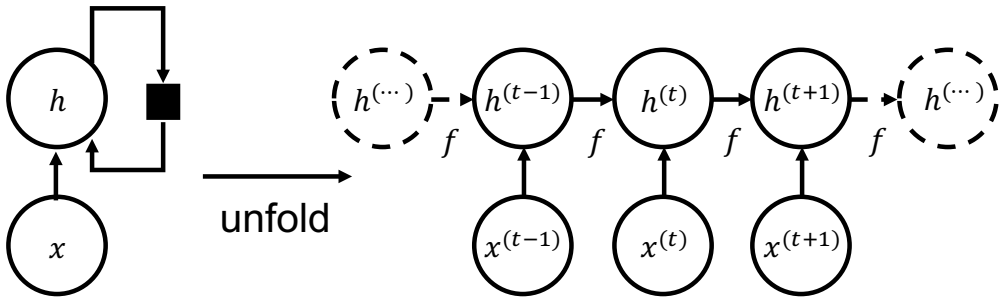
Example: Dynamic System Driven by External Signal

Unrolled computational graph



Example: Dynamic System Driven by External Signal

Unrolled computational graph

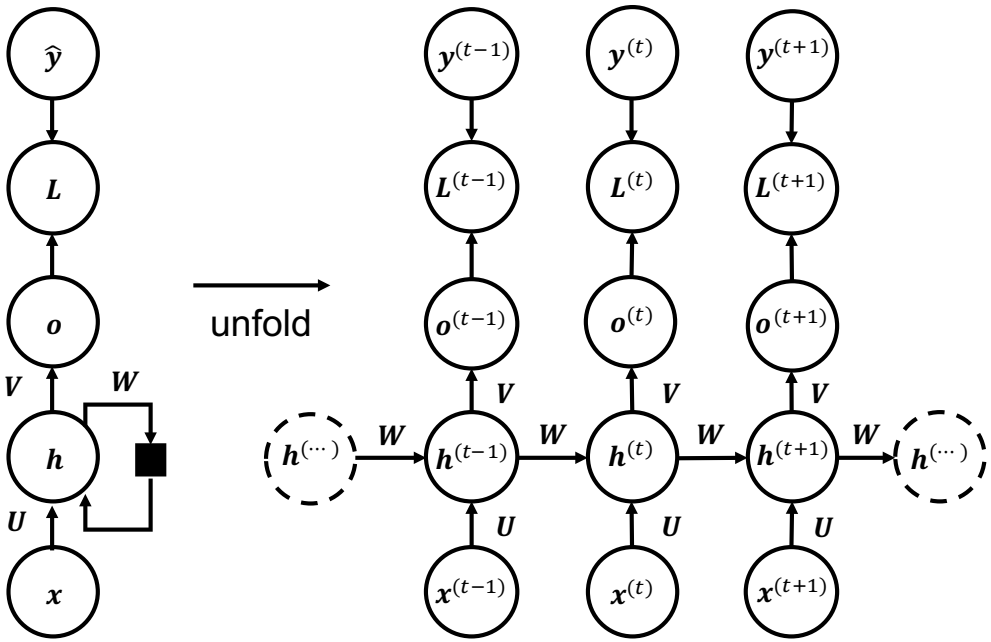


Interpretation of $\mathbf{h}^{(t)}$

- $\mathbf{h}^{(t)}$ acts a lossy summary of the past inputs up to time t
- Maps arbitrary length sequences, $\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \mathbf{x}^{(t-2)}, \ldots \mathbf{x}^{(2)}, \mathbf{x}^{(1)}$, to fixed length vector $\mathbf{h}^{(t)}$

Design Patterns

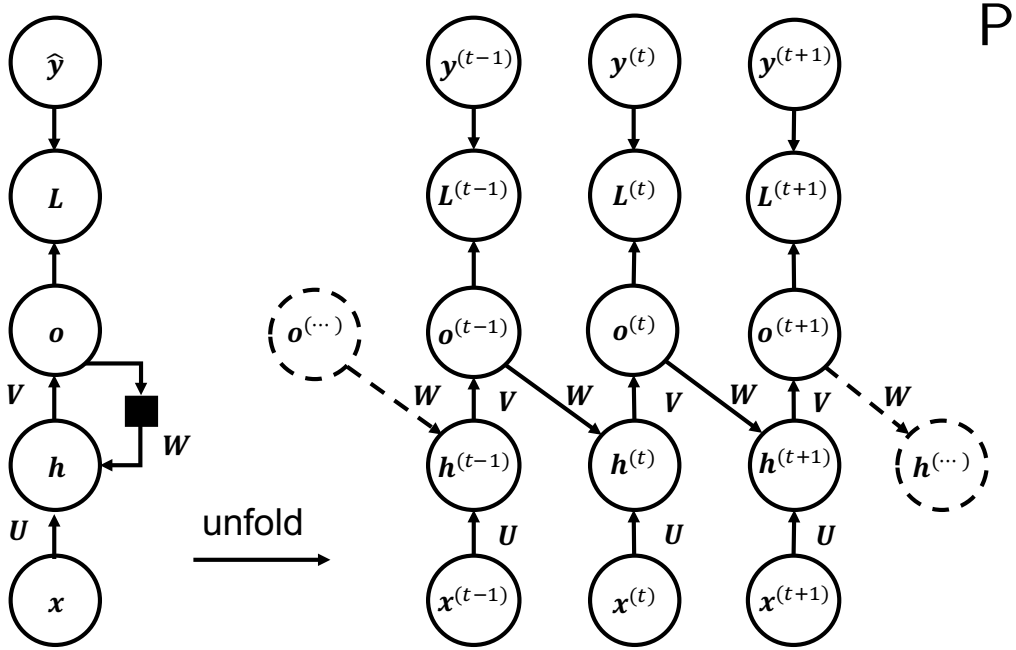
Produce an output at each time step, recurrent connects between hidden units



x := input sequence
o := output values
L := loss measure
y := training target
U := input-to-hidden connections weight matrix
W := hidden-to-hidden recurrent connections weight matrix
V := hidden-to-output connections weight matrix

Design Patterns

Produce an output at each time step, recurrent connections from the output at one time step to the hidden unit at the next time step



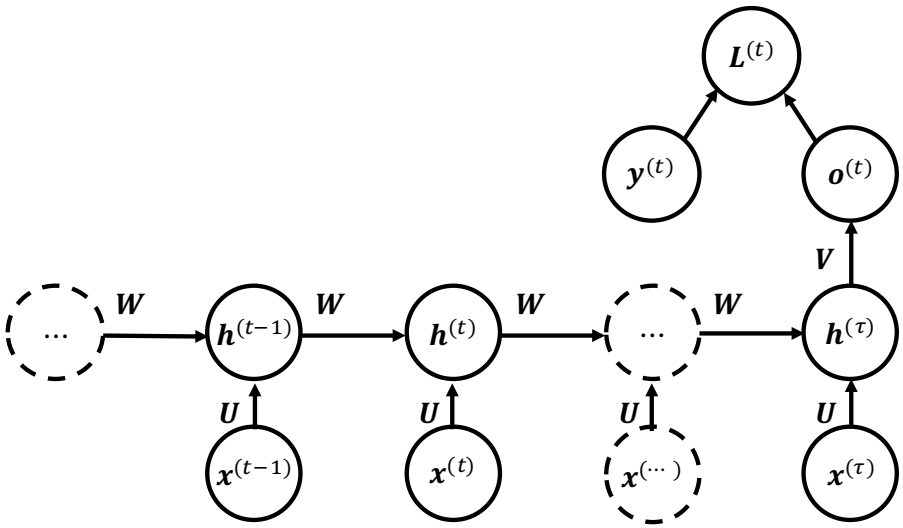
x: input sequence
$\mathbf{h}^{(t)}$: hidden layer activations
$\mathbf{o}^{(t)}$: outputs
$\mathbf{y}^{(t)}$: targets
$\mathbf{L}^{(t)}$: Loss

Properties

- Trained to put a specific output value into **o** and is the only information sent to the future
- No direct connections from **h** going forward
- Less powerful RNN but easier to train

Design Patterns

Recurrent connections between hidden units, read the entire sequence, produce a single output



- Can be used to summarize a sequence and produce a fixed-size representation used as input for further processing.
- Might have a target at the end
- Gradient on the output $\mathbf{o}^{(t)}$ can be obtained by back-propagating from the further downstream modules

Bidirectional RNN

Bidirectional RNNs

- RNNs have causal structure -¿ state at time t only captures information from the past
- Output prediction of the $y(t)$ which depends on the whole input sequence
- Successful in speech recognition, handwriting recognition, and bioinformatics

Bidirectional RNN

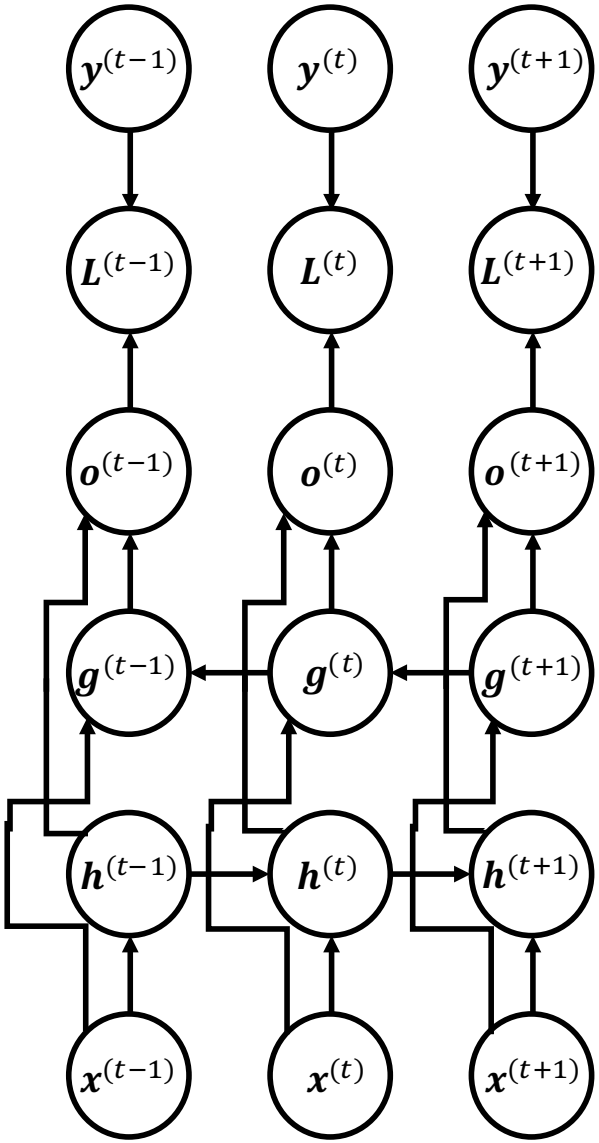
Bidirectional RNNs

- RNNs have causal structure -¿ state at time t only captures information from the past
- Output prediction of the $y(t)$ which depends on the whole input sequence
- Successful in speech recognition, handwriting recognition, and bioinformatics

Bidirectional RNN Structure

- Combines two RNNs, one moves that forward in time and one that moves backward in time
- Allows output units $\mathbf{O}(t)$ to compute representation that depends on past and future observations

Bidirectional RNN



x: input sequences

y: target sequences

$L^{(t)}$: loss at each time step t

h: recurrence that propagates information forward in time

g: recurrence that propagates information backwards in time

$\mathbf{o}^{(t)}$: at each time step t

Sequence-to-Sequence RNN

Overview

- Train RNN to map input sequence to output sequence of a different length
- Applications: Speech Recognition, machine translation, question answering
- Training and testing set not the same length

Sequence-to-Sequence RNN

Overview

- Train RNN to map input sequence to output sequence of a different length
- Applications: Speech Recognition, machine translation, question answering
- Training and testing set not the same length

Idea

1. Encoder/reader/input RNN processes $\mathbf{x} = \left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n_x)}\right)$ and emits the context $\mathcal{C}$, where $\mathcal{C}$ is a simple function of the RNNs final state
2. Decode/writer/output RNN generates the output sequence $\mathbf{y} = \left(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n_y)}\right)$

Sequence-to-Sequence RNN

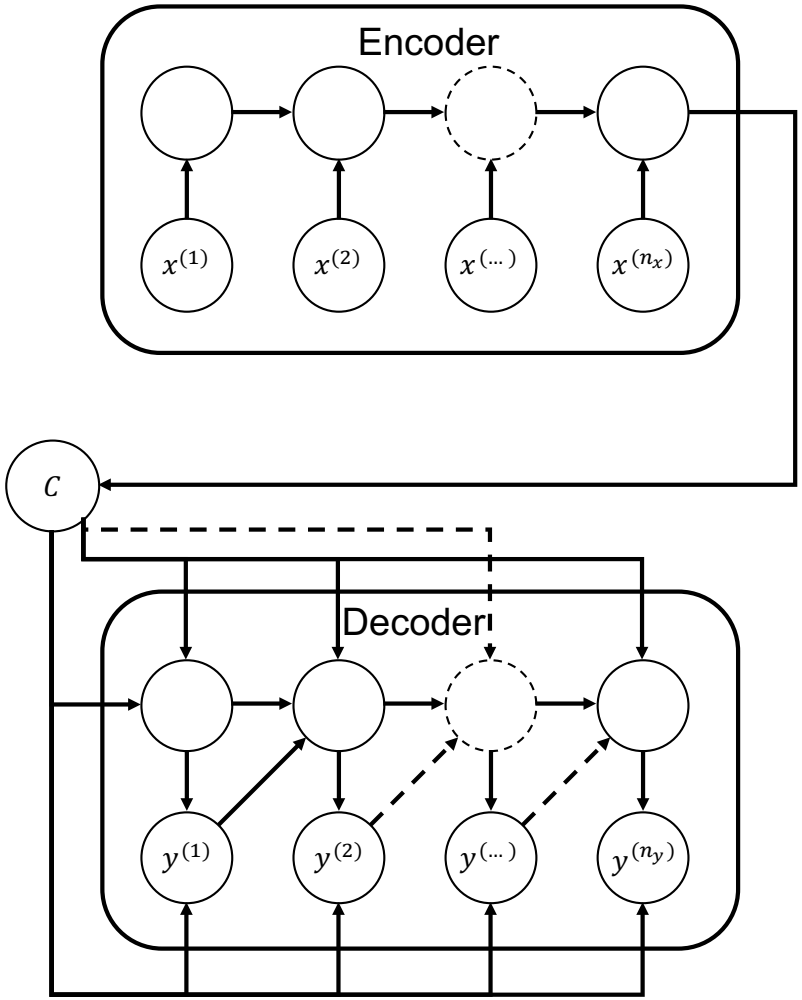
Sequence-to-sequence architectures are trained jointly to maximize the average of

$$\log P\left(\mathbf{y}^{(1)},\ldots,\mathbf{y}^{(n_y)}|\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(n_x)}\right) \tag{6}$$

Overall pairs of $\mathbf{x}$ and $\mathbf{y}$ sequences in the training set. Last state $\mathbf{h}_{nx}$

- Used as a representation $\mathcal{C}$ of the input sequence
- Input to the decoder RNN

Sequence-to-Sequence RNN



x: Input sequence

y: output sequence

- Encoder reads the input sequence
- Decoder generates the output sequence
- Final hidden state of the encoder is used to compute the fixed-size context variable $\mathcal{C}$
- Context $\mathcal{C}$ is used as the input the decoder

Long Term Dependencies

Challenges

- Gradients propagated over many stages tend to vanish or explode
- Exponentially smaller weights given to long-term interactions
- Recurrent networks involve composition of the same function many times

Long Term Dependencies

Challenges

- Gradients propagated over many stages tend to vanish or explode
- Exponentially smaller weights given to long-term interactions
- Recurrent networks involve composition of the same function many times

Recurrent Relation

- Function composition employed by RNN resembles matrix multiplication
- $\mathbf{h}^{(t)} = \mathbf{W}^T \mathbf{h}^{(t-1)}$
- Describes power relationship, .i.e. $\mathbf{h}^{(t)} = \left(W^{(t)}\right)^T \mathbf{h}(0)$

Long Term Dependencies cont.

Eigen decomposition of $\mathbf{W}$:

$$\mathbf{W} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T \tag{7}$$

Assume $\mathbf{Q}$ is orthogonal, the recurrence relation becomes

$$\mathbf{h}^{(t)} = \mathbf{Q}^T\mathbf{\Lambda}^t\mathbf{Q}\mathbf{h}(0) \tag{8}$$

Eigenvalues

- Eigenvalues are raised to power t
- Eigenvalues ¡ 1 decay to zero
- Eigenvalues ¿ 1 explode
- Any component of $\mathbf{h}(0)$ not aligned with the largest eigenvector will be discarded

Long Term Dependencies cont.

Avoiding RNN Vanishing Gradient Problem

- Stay in a region of parameter space where gradients do not vanish/explode
  - In order to store memories robust to small perturbations RNN must enter a region of parameter where gradients vanish/explode
- Uses other network architectures, i.e. Echo state networks and LSTMs

Echo State Networks

Most difficult things to learn in RNN

- Recurrent weights mapping from $\mathbf{h}^{(t-1)}$ to $\mathbf{h}^{(t)}$
- Inputs weights mapping $\mathbf{x}^{(t)}$ to $\mathbf{h}^{(t)}$

Echo State Networks

Most difficult things to learn in RNN

- Recurrent weights mapping from $\mathbf{h}^{(t-1)}$ to $\mathbf{h}^{(t)}$
- Inputs weights mapping $\mathbf{x}^{(t)}$ to $\mathbf{h}^{(t)}$

Echo State Networks (ESN) and Liquid State Machines (LSM)

- Set the recurrent weights such that the recurrent hidden weights capture the history of past inputs and only learn the output weights
- Known as reservoir computing
- Hidden units form temporal features, which capture different aspects of the history of inputs

Echo State Networks Cont.

Reservoir Computing

- Similar to kernel machines
- Map an arbitrary sequence into a fixed length vector
- Linear predictor applied to solve problem

Echo State Networks Cont.

Reservoir Computing

- Similar to kernel machines
- Map an arbitrary sequence into a fixed length vector
- Linear predictor applied to solve problem

Training

- Designed to be convex as a function of the output

Echo State Networks Cont.

Reservoir Computing

- Similar to kernel machines
- Map an arbitrary sequence into a fixed length vector
- Linear predictor applied to solve problem

Training

- Designed to be convex as a function of the output Weights
- Output consists of linear regression from the hidden units to the output weights
- Training criterion is mean squared

Echo State Networks Cont.

Reservoir Computing

- Similar to kernel machines
- Map an arbitrary sequence into a fixed length vector
- Linear predictor applied to solve problem

Training

- Designed to be convex as a function of the output Weights
- Output consists of linear regression from the hidden units to the output weights
- Training criterion is mean squared

How to Set Weights

- View recurrent net as a dynamical system
- Set the input and recurrent weights such that system is at the edge of stability

Multiple Time Scale Models

Objective

- Design a model that operates at multiple time scales
- Fine grained time scales $\rightarrow$ small details
- Coarse time scales $\rightarrow$ transfers information from the past to the present efficiently.

Multiple Time Scale Models

Objective

- Design a model that operates at multiple time scales
- Fine grained time scales $\rightarrow$ small details
- Coarse time scales $\rightarrow$ transfers information from the past to the present efficiently.

Fine and Coarse Model Strategies

- Skip connections across time
- Leaky units that integrate signals with different time constants
- Remove some connections used to model fine-grained time scales

Adding Skip Connections Through Time

Idea

- Technique or coarse time scales
- Add direct connections from variables in the distant past to variables in the present
- Allows for capturing longer dependencies

Adding Skip Connections Through Time

Idea

- Technique or coarse time scales
- Add direct connections from variables in the distant past to variables in the present
- Allows for capturing longer dependencies

Gradients

- Vanish/explode exponentially w.r.t the number of time steps τ
- Mitigate with recurrent connections with time delay of d
- Gradients diminish exponentially as a function of τ/d rather than τ

Leaky Units and a Spectrum of Different Time Scales

Idea

- Units with linear self-connections and weight near one

Leaky Units and a Spectrum of Different Time Scales

Idea

- Units with linear self-connections and weight near one

Leaky Units

- Accumulate a running average of $\mu^{(t)}$ of some value $v^{(t)}$ by applying the update

$$\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1-\alpha)v^{(t)} \tag{9}$$

- α is a linear self-connection from $\mu^{(t-1)}$ to $\mu^{(t)}$
- α near one $\rightarrow$ remembers information about the past for a long time
- α near zero $\rightarrow$ information about the past is rapidly discarded

Leaky Units and a Spectrum of Different Time Scales

Benefits

- Ensures a unit can always learn to be influenced by a value d time steps earlier
- Can use a linear self-connection with a weight near one
- Allows for smoother adaptation and flexibility by adjusting the real valued alpha instead of the skip length

Leaky Units and a Spectrum of Different Time Scales

Benefits

- Ensures a unit can always learn to be influenced by a value d time steps earlier
- Can use a linear self-connection with a weight near one
- Allows for smoother adaptation and flexibility by adjusting the real valued alpha instead of the skip length

Strategies for setting the time constant

- Manually fix them
- Make them a free parameter and learn them, helps with long term dependencies

Removing Connections

Idea

- Organize the state of the RNN at multiple time scales
- Remove length one connections and replace them with longer connections
- Modified units are forced to operate at long time scales, unlike skip connections may focus on short-time scales

Removing Connections

Idea

- Organize the state of the RNN at multiple time scales
- Remove length one connections and replace them with longer connections
- Modified units are forced to operate at long time scales, unlike skip connections may focus on short-time scales

Ways to force units to operate at different time scales

- Make units leaky $\rightarrow$ different groups of units associated with different fixed time scales
- Have explicit and discrete updates at different times with different frequency for different groups

LSTM and Gated RNNs

LSTM and Gated RNNs

- The most effective sequence models for practical applications
- Have paths through time that neither vanish nor explode
- Gated RNN weights may change at each time step

LSTM

LSTM

- Uses self-loops to produce paths where the gradient can flow for long durations
- Weights on the self loop are conditioned on context rather than fixed.
- Time scale of integration can be changed dynamically
- Time scale of integration is changed on the input sequence

Applications of LSTM

- Unconstrained handwriting recognition
- Speech recognition
- Handwriting generation
- Machine translation
- Image captioning
- Parsing

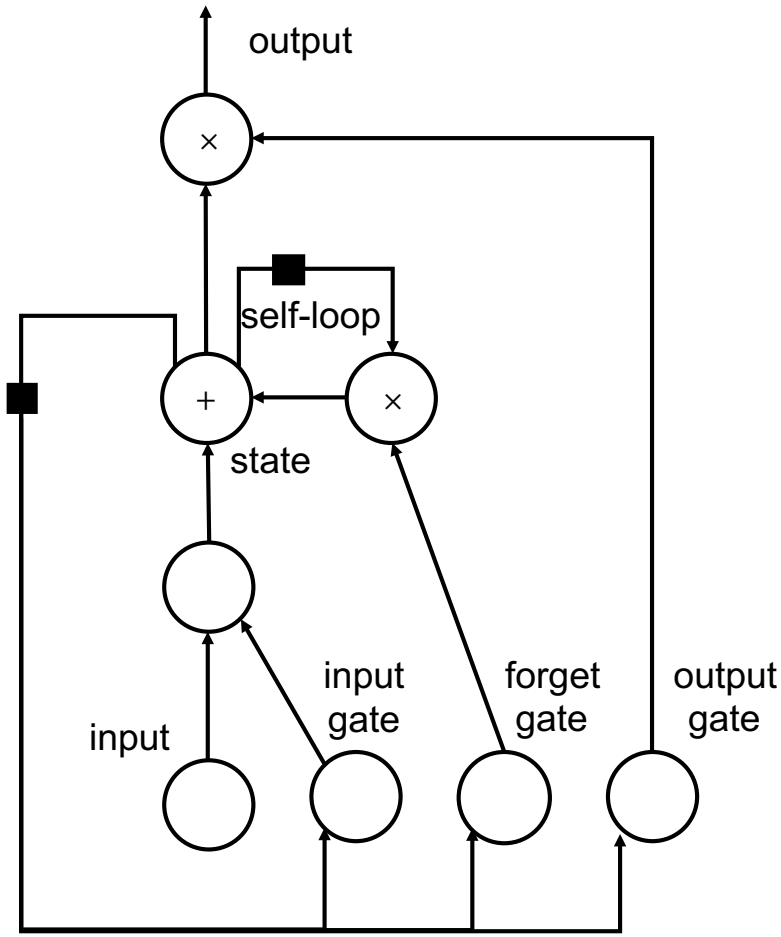
LSTM

Basic LSTM

- Applies element-wise non-linearity to the affine transformation of inputs and recurrent units

LSTM "cells"

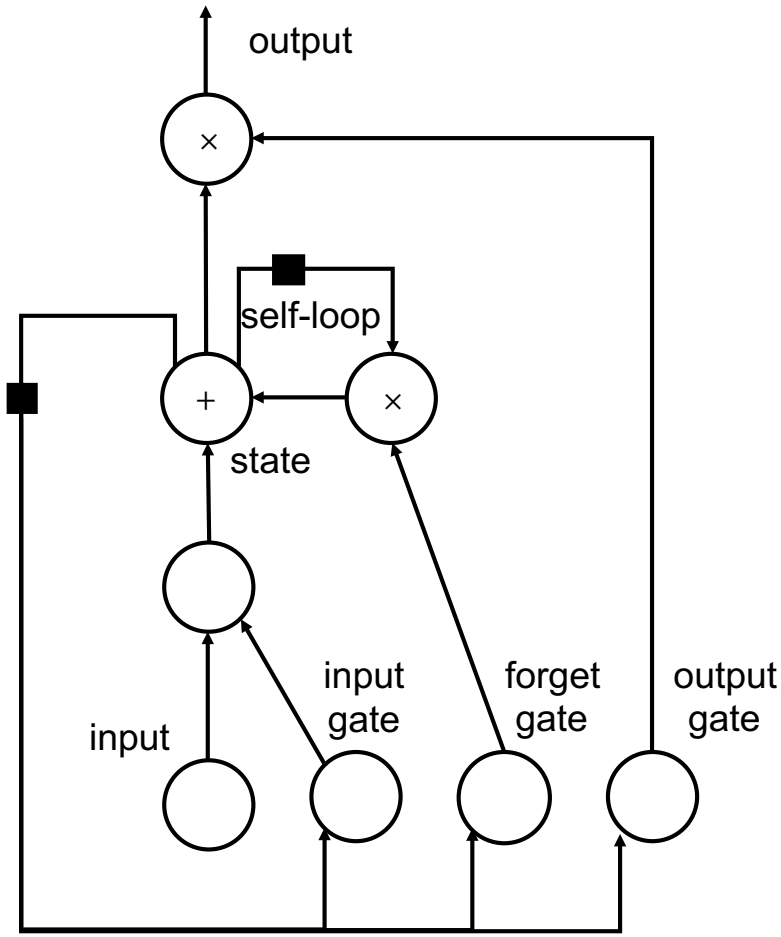
- Cells are connected recurrently to each other
- Cells replace hidden units of recurrent networks
- Input feature computed with a regular artificial neuron unit
- Values can be accumulated into state if input gate allows it



LSTM

LSTM "cells"

- State unit has a linear self-loop whose weight is controlled by output gate
- Output of cell can be shut off by the output gate
- All gating units have a sigmoid nonlinearity
- State unit can be used as an extra input to the gating units



Application

Google Neural Machine Translation System

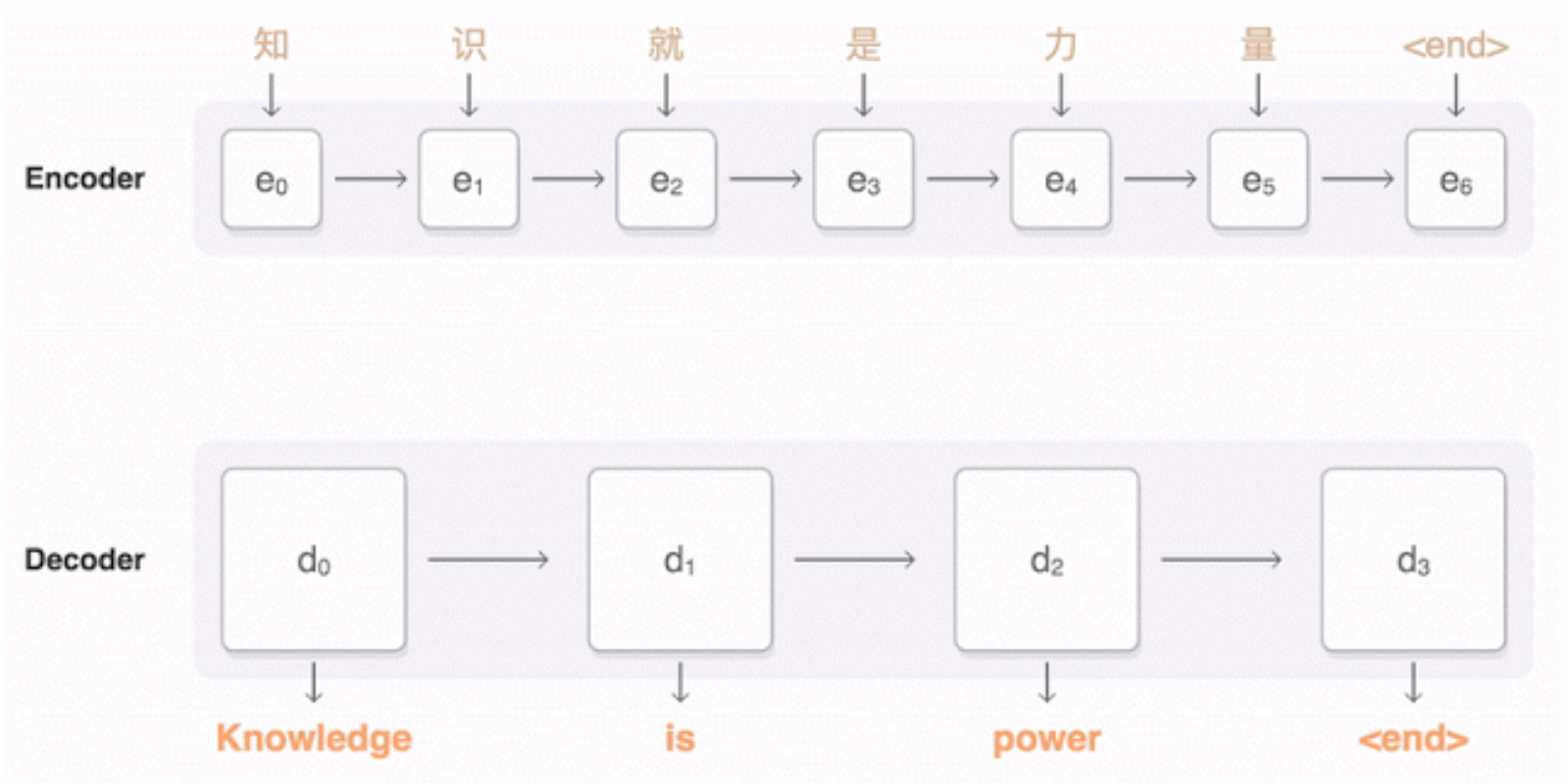
- Google translate uses a phrase-based machine translation (PBMT) based algorithm
- Uses recurrent neural networks
- Sequence-to-sequence RNN

Sequence-to-sequence RNN vs PBMT

- Sequence-to-sequence RNNs directly learn the mapping from the input phrase to the output phrase
- PBMT breaks input sentence into words and phrases to be translated independently.
- Neural machine translation (NMT) requires fewer engineering design choices
- Initially, NMT showed equivalent accuracy as PBMT

Application

- Network first encodes the Chinese works as a list of vectors
- Once the entire sentence is read the decoder begins generating the English sentence one word at a time



Application

- GNMT reduces translation errors by more than 55% - 85% on several major language pairs