Machine Learning 410

Lesson 5

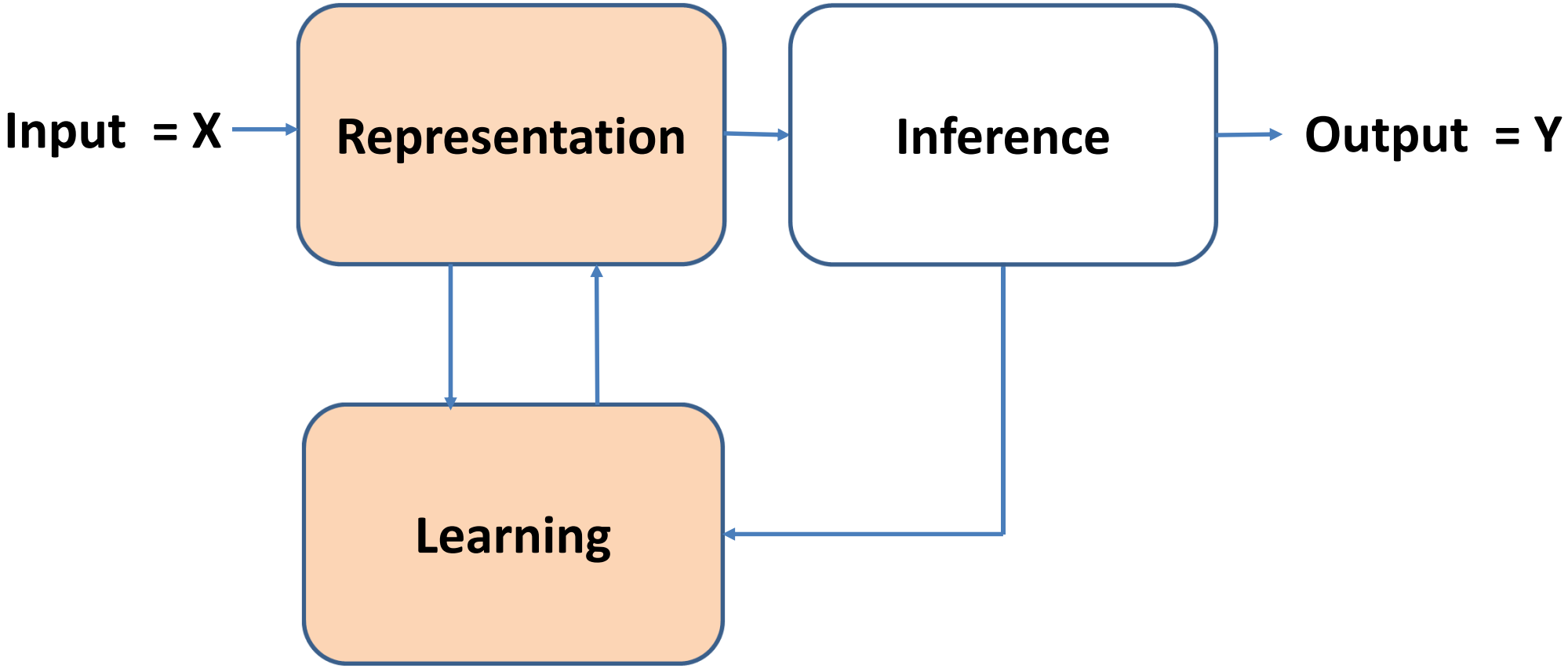
Introduction to Recurrent Neural Networks

Steve Elston

Reminders

- Discussion – get an easy 5 points!
  - Discussion 4 closes Tonight, Oct 24.
  - Discussion 5 closes Thursday, Oct 31.
- Homework – **Updated homework is in Canvas**
  - Homework 3 due Oct 27
  - Homework 4 due Nov 3
- Materials in course Github repository
  - Notebook with reading
  - Slides

Recurrent Neural Networks



Recurrent Neural Networks

- Many data types are **ordered sequences**:
  - Numerical time series
  - Natural language
  - Speech
- **Recurrent neural networks** operate on ordered sequences
- Recurrent architecture first proposed by Rumelhart et. al. (1986)
- The Long Short Term Memory (LSTM) network proposed by Hochreiter and Schmidhuber (1997)
  - LSTM and GRU networks now dominate speech recognition applications

Recurrence for Neural Networks

- Recurrent neural networks are neural networks that learn **recurrent functions**
- A recurrent function is a function that operates on itself
- Consider a simple function:

$$s^{(t)} = f(s^{(t-1)}; \theta)$$

- The first recurrence step yields:

$$s^{(t+1)} = f(f(s^{(t-1)}; \theta); \theta)$$

- Recurrence can continue indefinitely

Recurrence for Neural Networks

- Same function and weights are used for each time step
  - Only **valid for stationary data**
  - **Distribution** of stationary data is **constant in time**
- Recurrence **adds memory** to the neural network
- The recurrence relationship is **causal**; operates in time order
- The length of the memory grows with each time step
  - Time scale changes with time
  - Can be problem with long sequences/memory
- Memory from step to step; is **non-Markov** behaviour
  - 1st order **Markov process** is stochastic process with **no memory**

Basic RNN architecture

Loss function

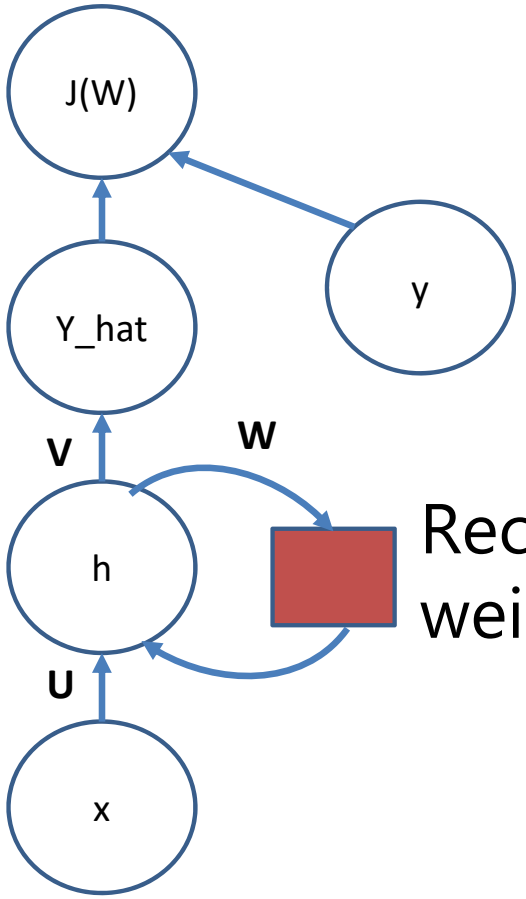
Output layer

Output weight tensor

Hidden layer

Input weight tensor

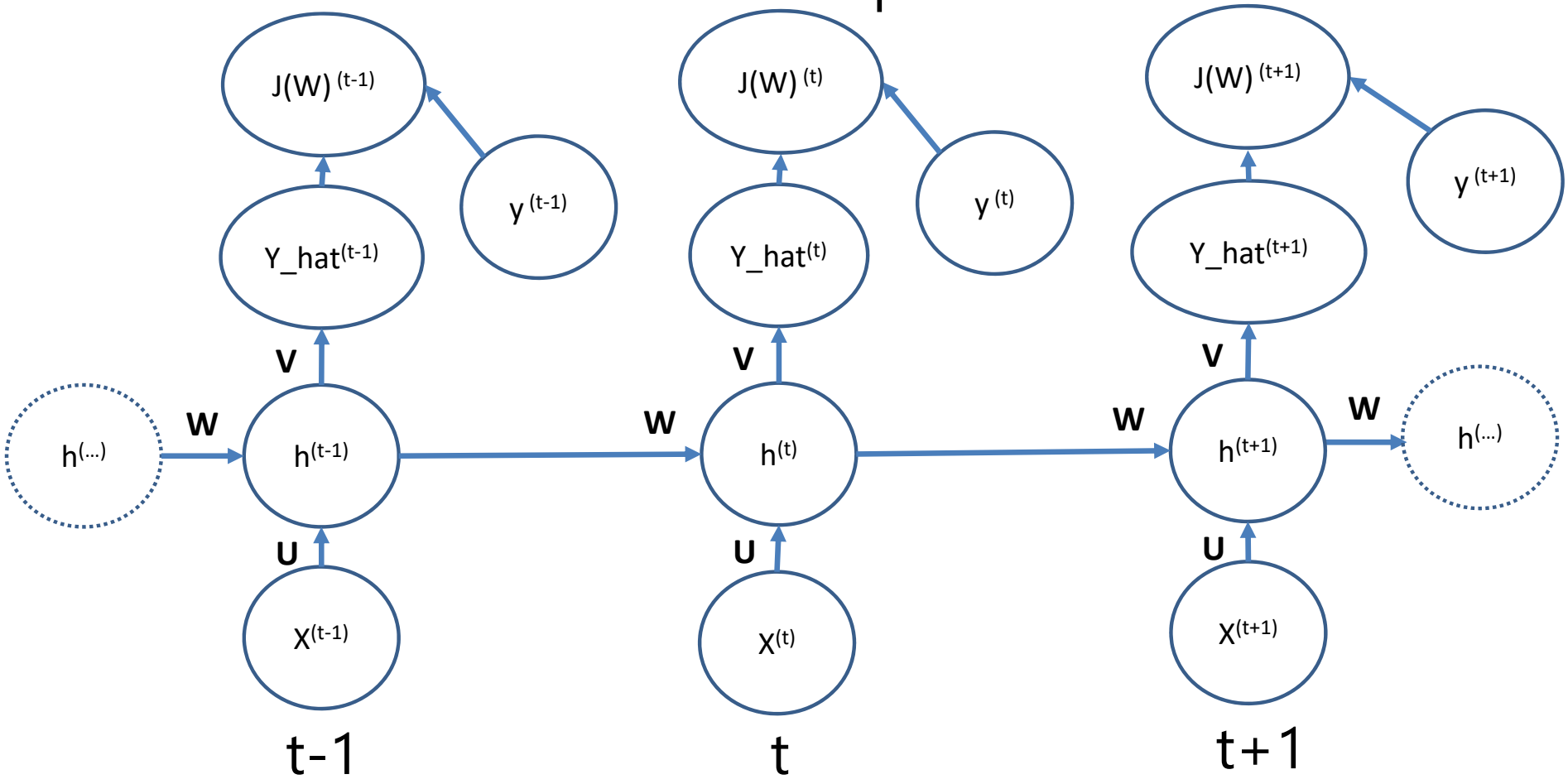
Input layer



Recurrence with weight tensor W

Basic RNN architecture

Unfold the recurrence relationship in time:



Basic RNN architecture

- Forward propagation in RNNs

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$ Value of recurrent layer

$$h^{(t)} = tanh(a^{(t)})$$ Activation of recurrent layer

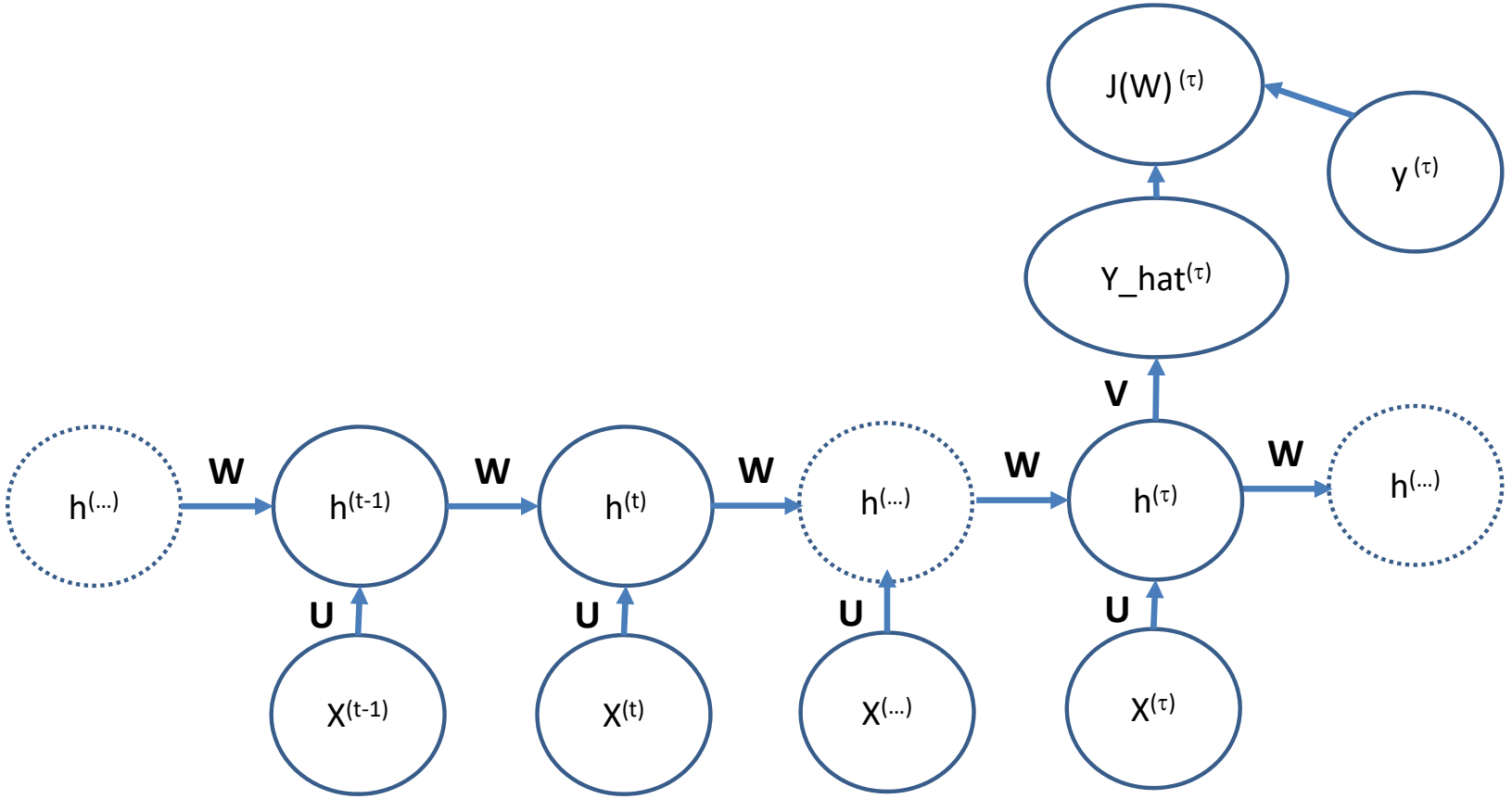
$$o^{(t)} = c + Vh^{(t)}$$ Value of output layer

$$\hat{y}^{(t)} = softmax(o^{(t)})$$ Activation of output layer

- RNNs use shared weight tensors
- **Learn weights** of RNNs using **backpropagation through time**

Single output RNN architecture

Single output RNN used to detect a particular sequence

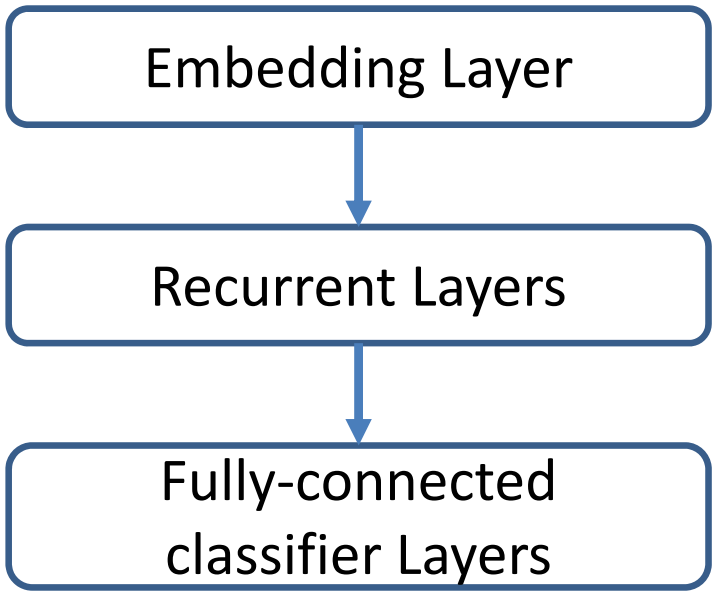


Single output RNN architecture

- Single output RNN is difficult to train
- Backpropagaton though time requires a gradient of the loss function
  - But single output RNN only produces value once
  - Can encounter very slow training

Embedding and Natural Language Processing

- Embedding models create lower dimensional feature map for **natural language processing (NLP)**
- Embedding layer creates feature map for RNN



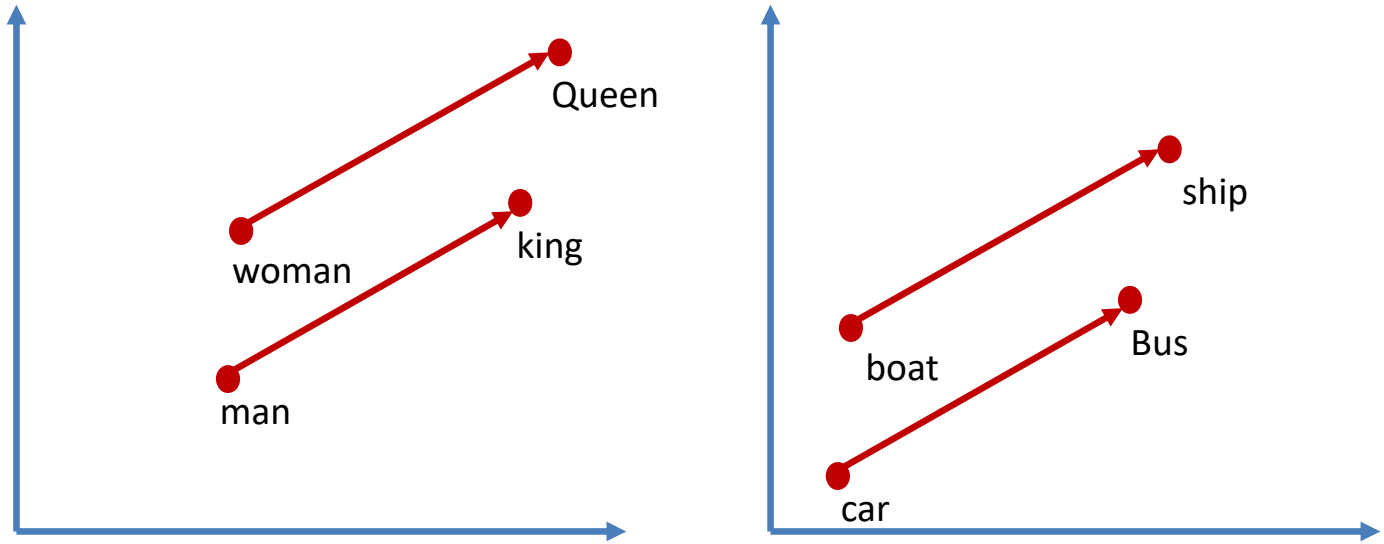
Creates feature map

Sequential model of features

Classify document

Embedding and Natural Language Processing

- Embedding maps sparse **Token** space (typically words) to a dense lower dimensional feature space
- Embedding models measure similarity



- Unfortunately, unclear how embedding models work

The Bag of Words Model and Text Preparation

Some NLP terminology:

- A **corpus** is comprised of **documents**
  - Document can be email, news article, Tweet, book, paragraph, etc.
- A document is comprised of **tokens**
  - Tokens can be words, sequences of words, etc.

The Bag of Words Model and Text Preparation

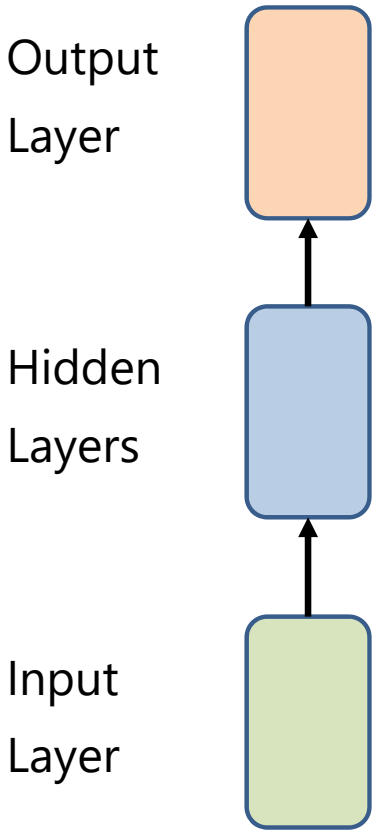
- The **bag of words model,** or **BOW model**, is a basic NLP model
- BOW assumes **exchangeability of words**
  - Order does not matter
  - Independent of grammar
- BOW based on **count of tokens**
  - **Term document matrix (TDM)**; words in rows, documents in columns
  - **Document term matrix (DTM)**; documents in rows, words in columns
  - TDM and DTM are very sparse

The Bag of Words Model and Text Preparation

- Preparing text for analysis
- **Normalize** text
  - All characters to lower case
  - Remove punctuation
  - Remove numbers and special characters
- **Stop words** are removed
  - Stop words are common words with no semantic value
  - Examples; ‘and’, ‘the’, ‘or’
  - Choice of stop words can be domain dependent
- Document is **tokenized**
  - Token are typically words or sequence of words

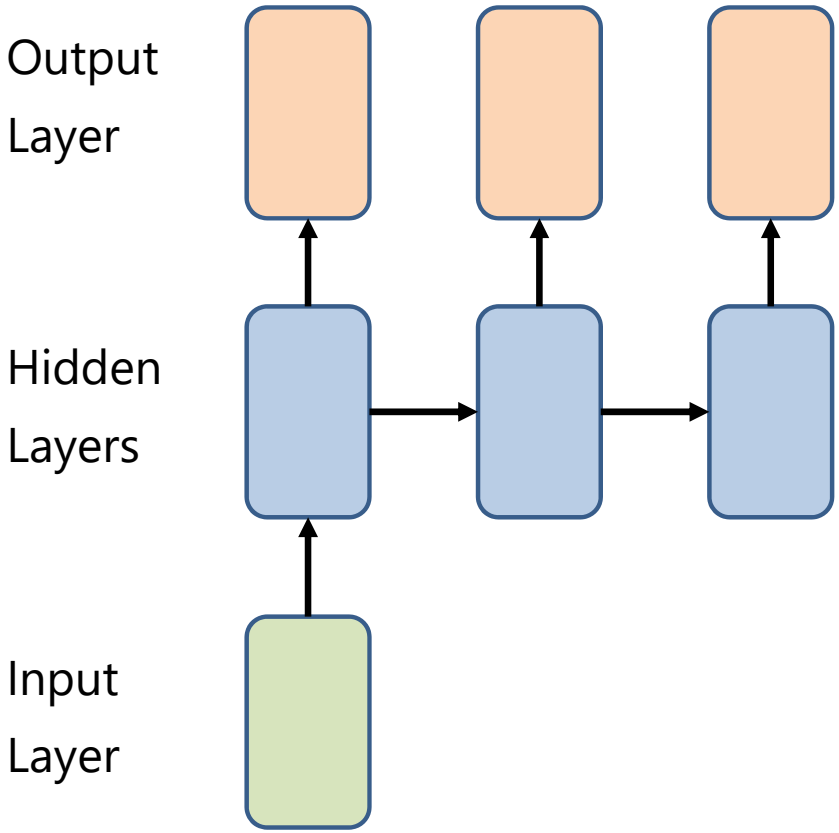
RNN Architectures

Feedforward network



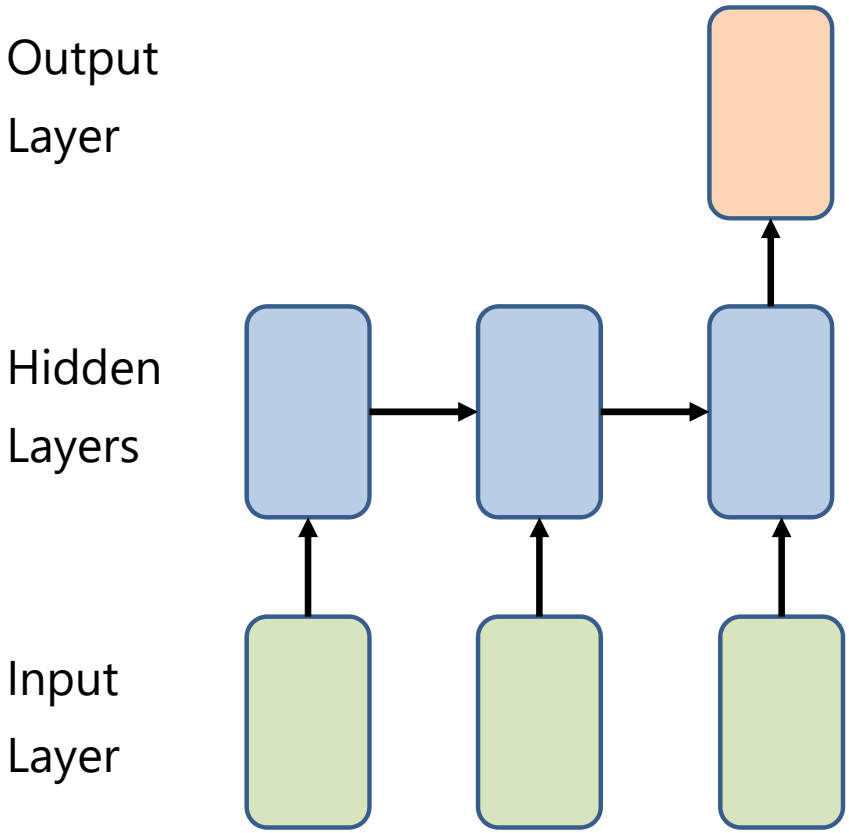
RNN Architectures

Generative model where input vector generates output sequence



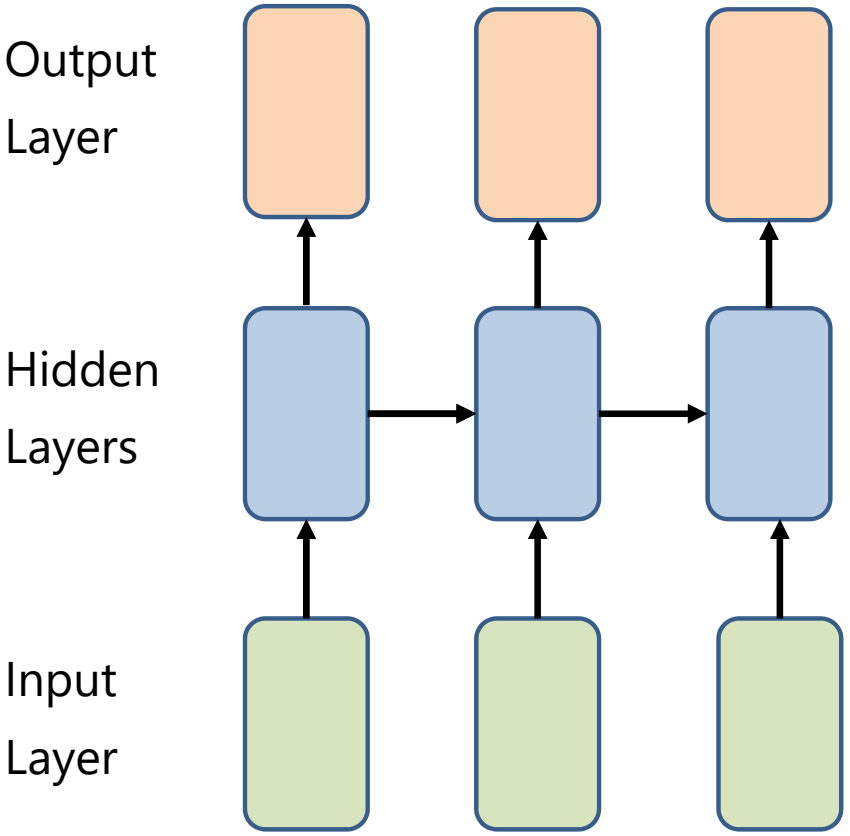
RNN Architectures

Classification of input sequence



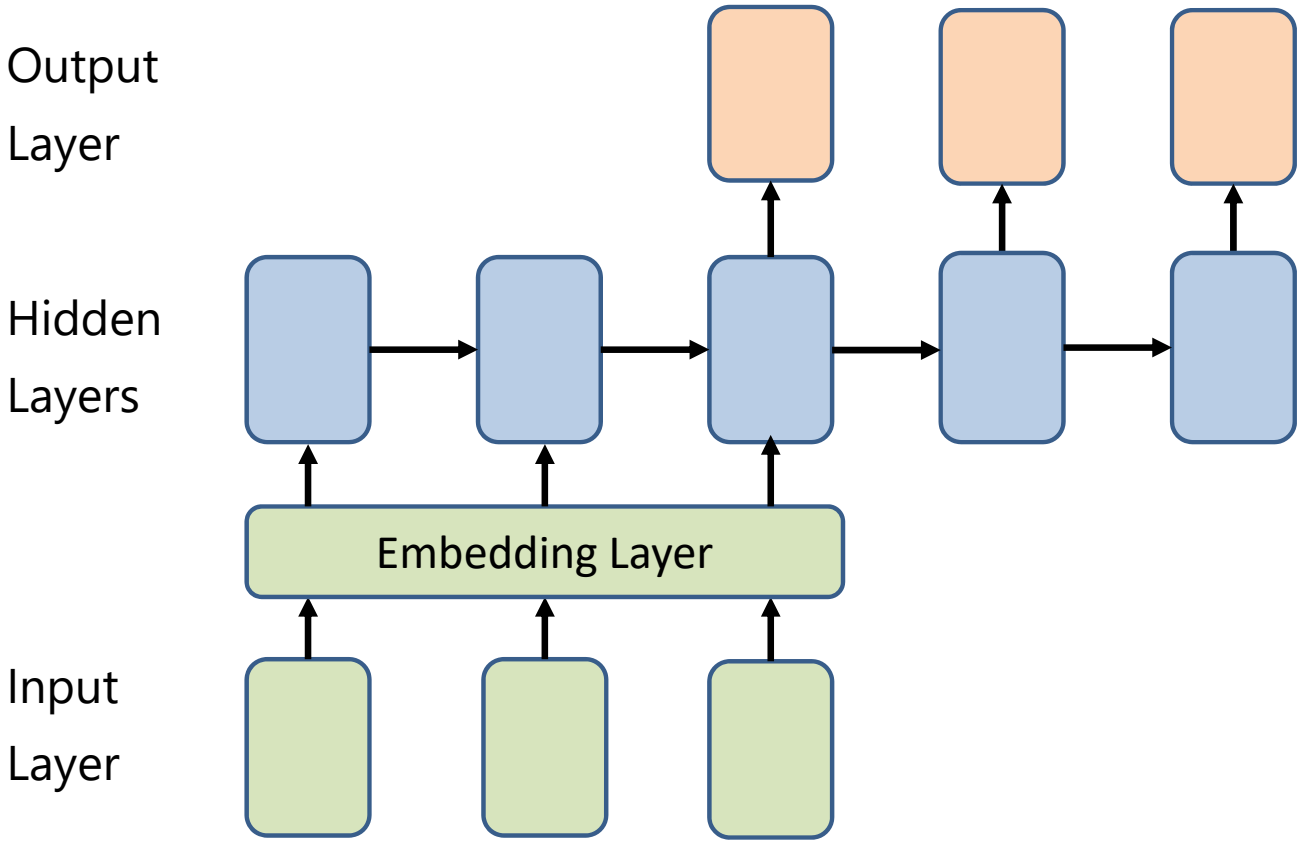
RNN Architectures

Sequence-to-sequence model – Generative



RNN Architectures

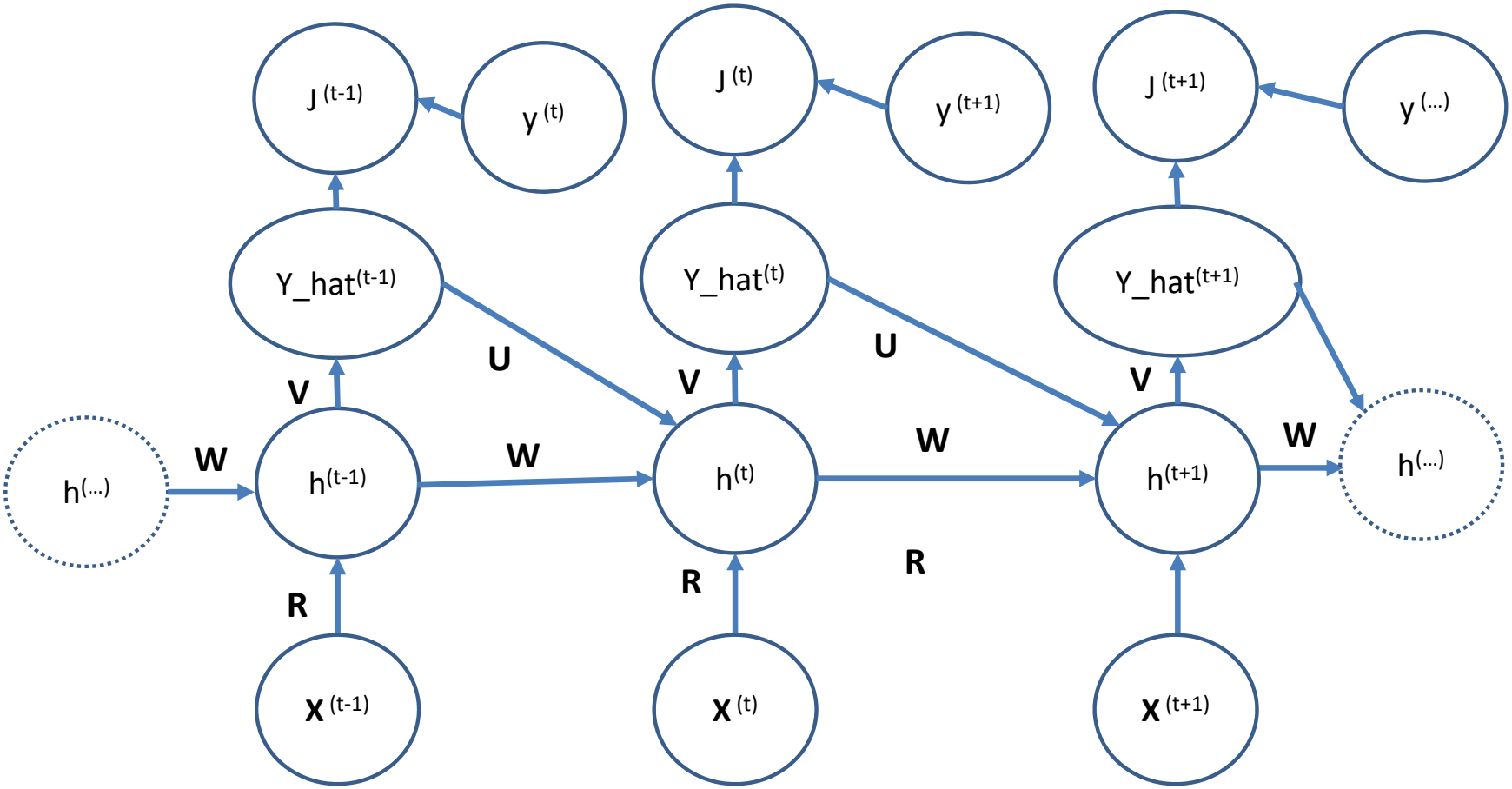
Sequence-to-sequence model with **context** and embedding



Sequence Generation with RNNs

- Given an input value **x**, we want to generate an output sequence **y**.
  - Activation is function of input and last output: $Y_t = h_{t-1}(y_{t-1},x)$
  - Is a generative model
- Train by minimizing the loss with respect to the desired response
  - Loss function = J(U, V, W)
- Applications:
  - Response to question – chat bots
  - Caption images
  - Machine translation

Sequence Generation with RNNs



Adding Depth to RNNs

- Single recurrent layer has limited capacity
- Adding additional recurrent layers adds capacity to the model
- Adding depth to RNNs presents training problems
  - Vanishing and exploding gradients

Adding Depth to RNNs

Output layer

Output weight tensor

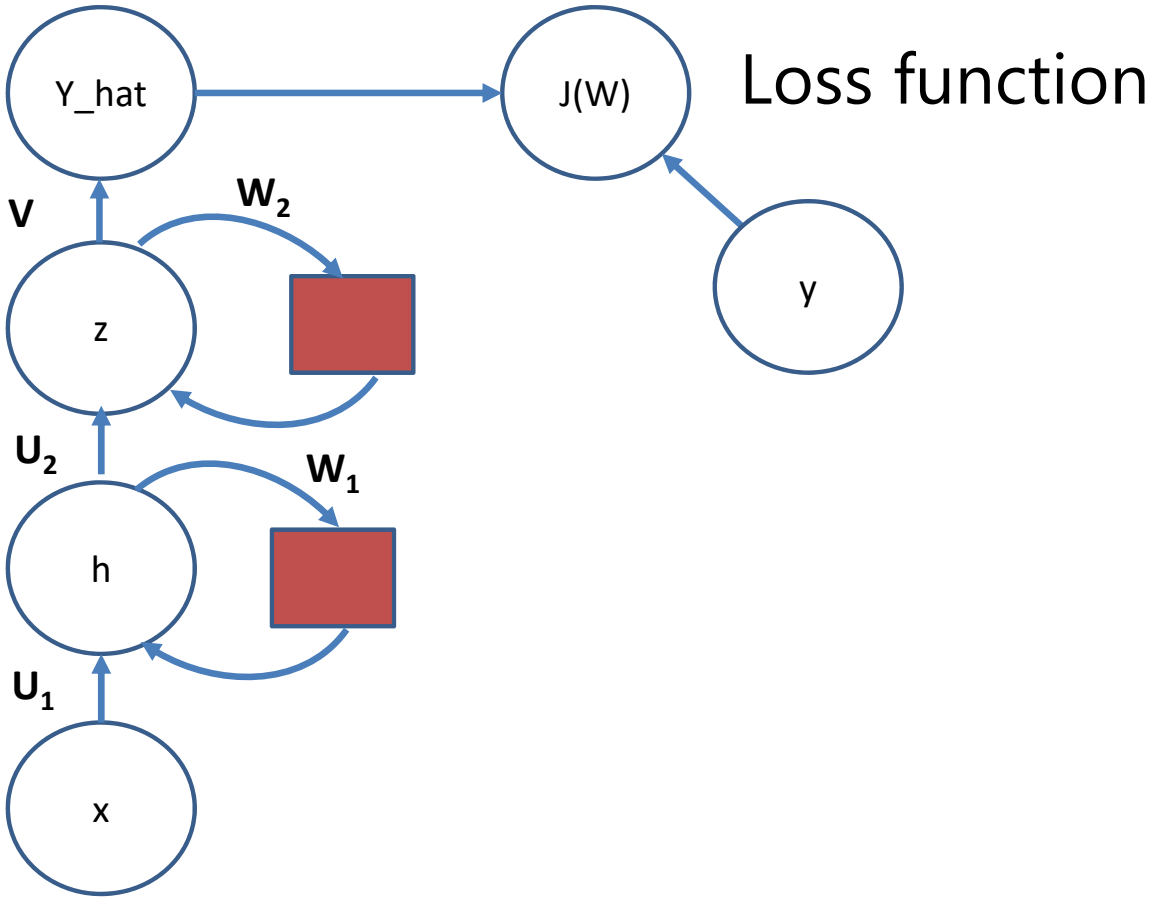
Hidden recurrent layer

Hidden weight tensor

Hidden recurrent layer

Input weight tensor

Input layer



Adding Depth to RNNs

- Training multiple recurrent layers can be problematic
  - Recurrence layers introduce delay
  - Delay affects the gradient
  - The loss function is slow to update
- A possible solution is to add a **skip loop** to the recurrent layer
  - A skip loop is a recurrence loop with memory
  - The skip loop forces the recurrence relationship to be sensitive at each time step

Adding Depth to RNNs

Output layer

Output weight tensor

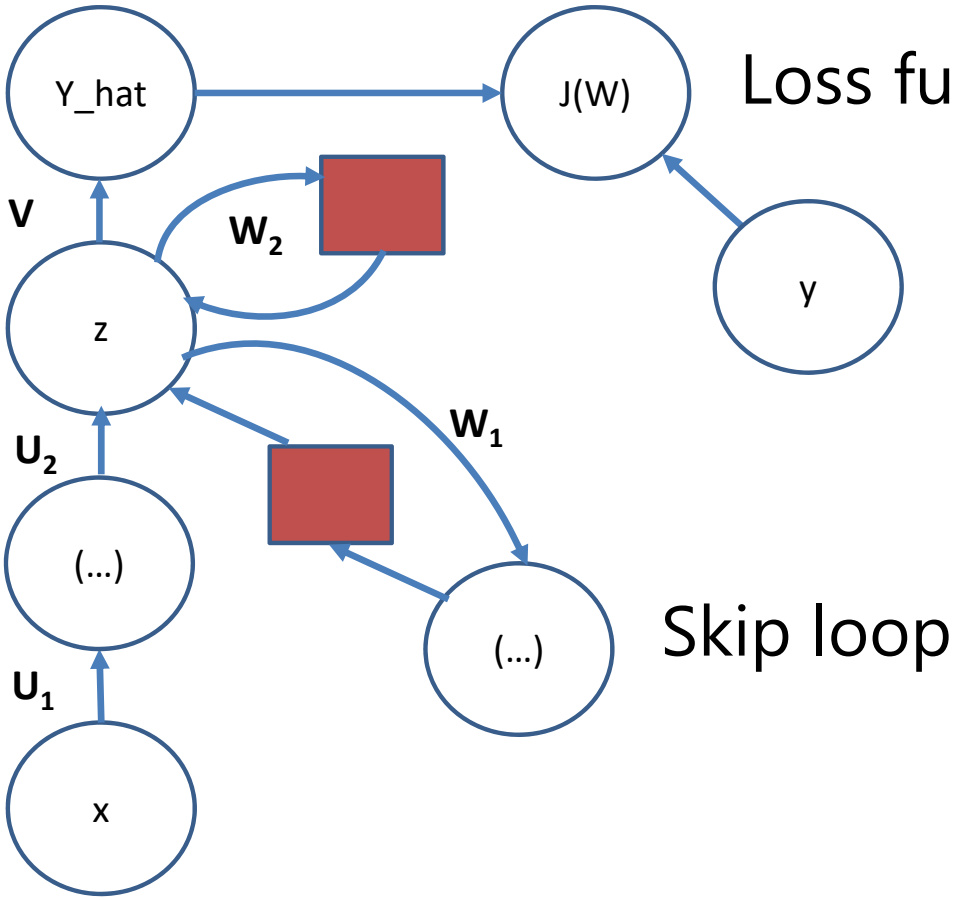
Hidden recurrent layer

Hidden weight tensor(s)

Hidden layer(s)

Input weight tensor

Input layer



Long-Short Term Memory

- Problems training simple recurrent architectures have led to the development of better approaches
  - Vanishing and exploding gradients common
  - Consider the following recurrent relationship:

$$s^{(t+n)} = f(f(\ldots f(f(s^{(t)};\theta));\theta)$$

For a long recurrence, large n:

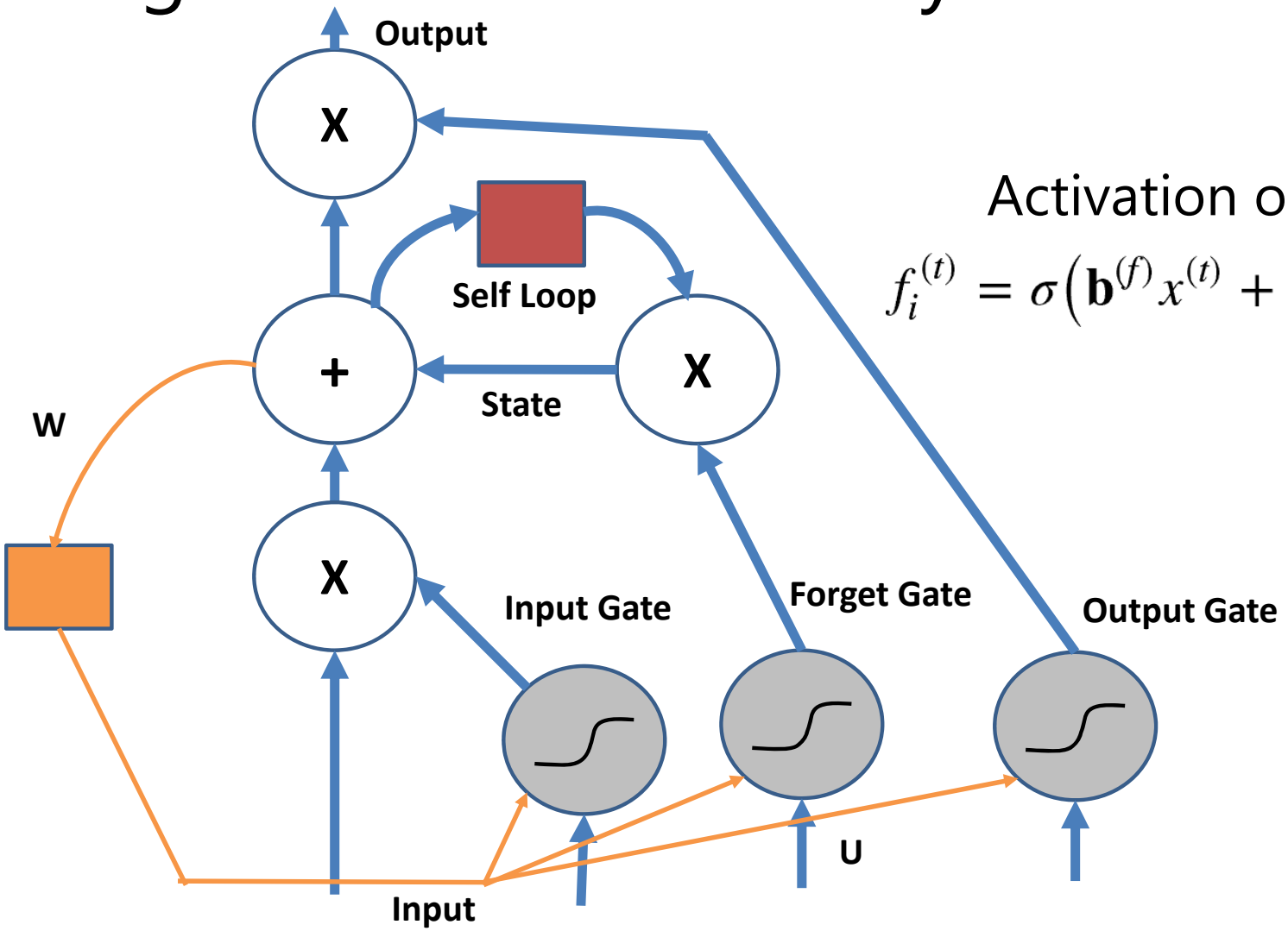
If $f(s;\Theta) > 1.0$ the gradient grows exponentially

If $f(s;\Theta) < 1.0$ the gradient vanishs

Long-Short Term Memory

- How can one create a neural net for modeling sequences with stable gradients?
  - **Memory** lets the NN operate at multiple time scales
  - **Forget gates** break the recurrence relationship and stabilize the gradient
- The **Long-Short Term Memory (LSTM)** neural network was an early architecture using memory and forget gates
- LSTM used for speech recognition, handwriting generation, machine translation, etc.

Long-Short Term Memory



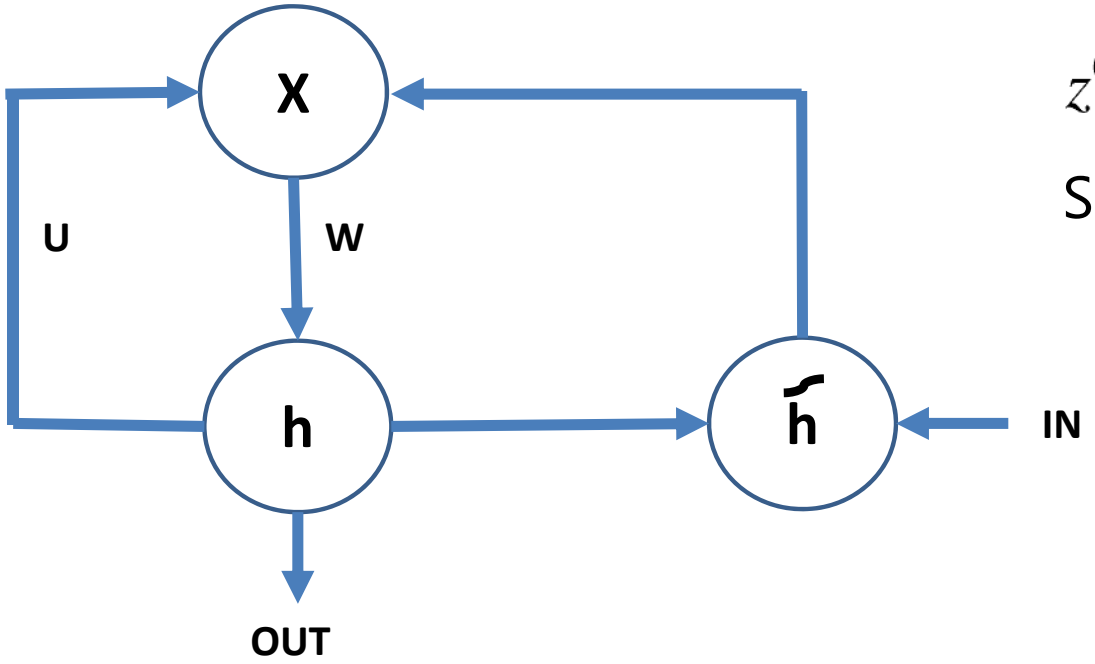
Activation of forget gate

$$f_i^{(t)} = \sigma\left(\mathbf{b}^{(f)} x^{(t)} + \mathbf{U}^{(f)} x^{(t)} + \mathbf{W}^{(f)} h^{(t-1)}\right)$$

Gated Recurrent Units

- LSTM networks are effective but a bit complex
- Is there a simpler approach?
- The **Gated Recurrent Unit (GRU)** architecture (Cho et. al., 2014)
  - Operates on multiple time scales
  - No memory units
  - Greatly simplifies the architecture
- GRUs are widely applied in speech recognition, machine translations, etc.

Gated Recurrent Units



Output determined by an exponential decay function

$$h^{(t)} = \left(1 - z^{(t)}\right) h^{(t-1)} + z^{(t)} \tilde{h}^{(t-1)}$$

$$z^{(t)} = \sigma\left(U_{(z)} h^{(t-1)} + W_{(z)} x^{(t)}\right)$$

Stabilizes the gradient

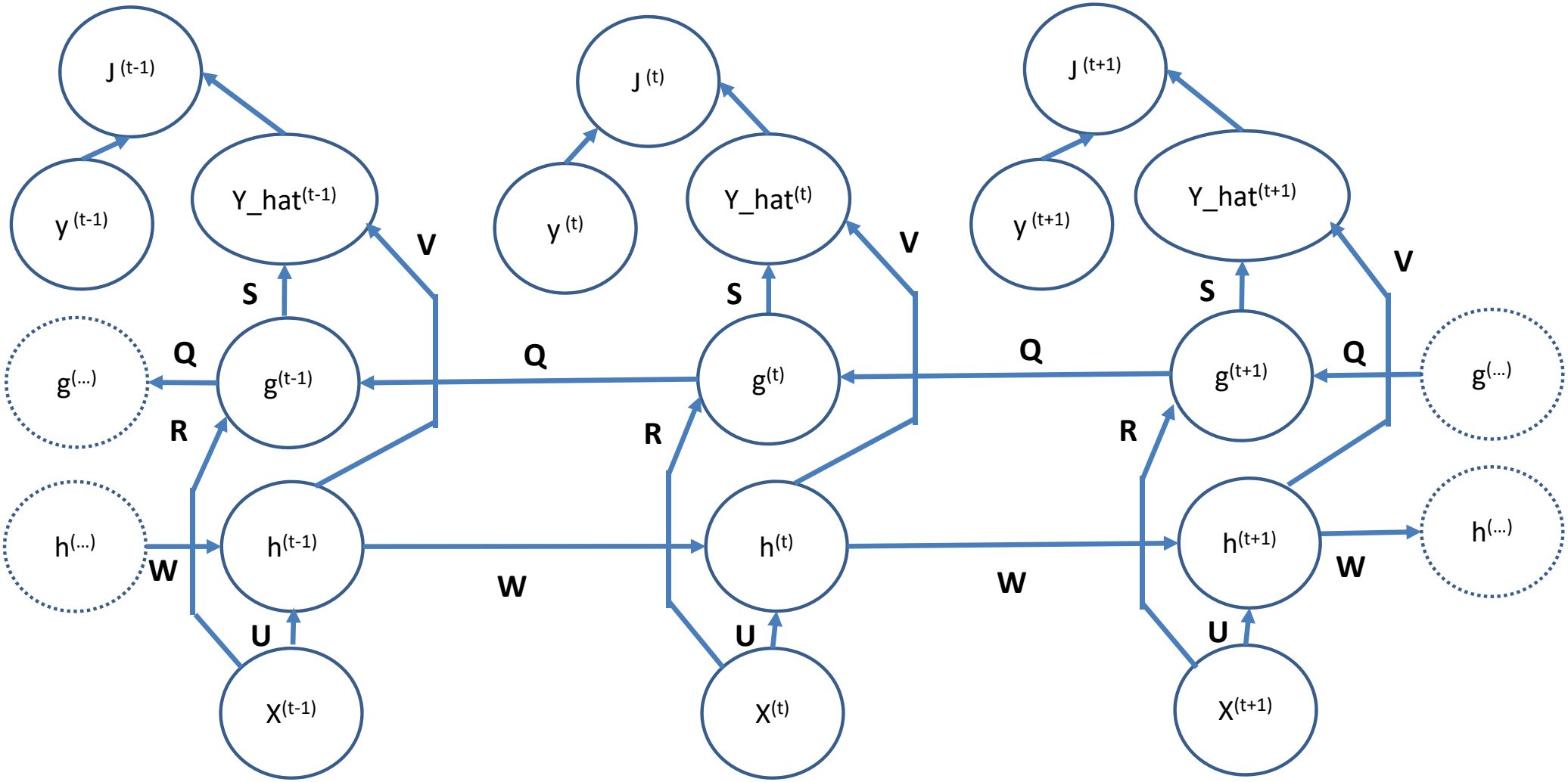
Regularization for RNNs

- How can you regularize RNN training?
- l1 and l2 regularization work
- Use modified recurrent dropout regularization
  - Standard dropout regularization interferes with BPTT
  - **Recurrent dropout regularization** does not Bernoulli sample on each forward pass
- Add a batch normalization layer to the recurrent layer

Bidirectional RNNs

- Not all sequential relationships are causal
  - A natural language phrase can be parsed in both directions
  - Hand writing recognition can proceed from either end
  - Figure captioning has no preferred direction
- For non-causal sequences we can use **bidirectional RNNs**
- Bidirectional RNNs trained using BPTT in both directions

Bidirectional RNNs



Bidirectional RNN Architectures

Bidirectional sequence-to-sequence model with **context** and embedding