Recurrent Neural Networks

CSCI 447/547 MACHINE LEARNING

Outline

- Introduction
- Sequence Data
- Sequential Memory
- Recurrent Neural Networks
- Vanishing Gradient
- LSTMs and GRUs

Introduction

- Uses:
  - Speech Recognition
  - Language Translation
  - Stock Prediction
  - Video
  - Weather
- Incorporate internal memory
- Used when “temporal dynamics that connects the data is more important that the spatial context of an individual frame” (Lex Fridman, MIT)

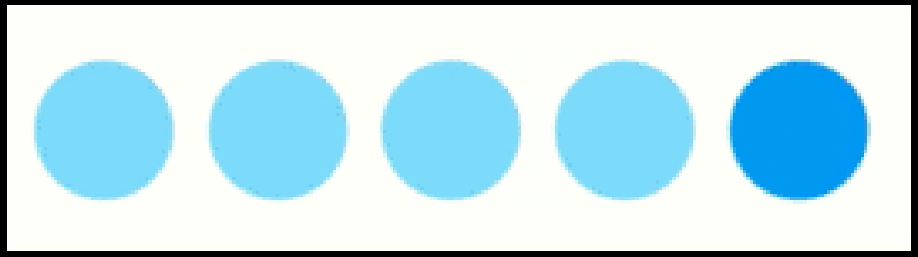
Sequence Data

- Snapshot of a ball moving in time:

- You want to predict the direction it is moving
  - With the data you have, it would be a random guess

Sequence Data

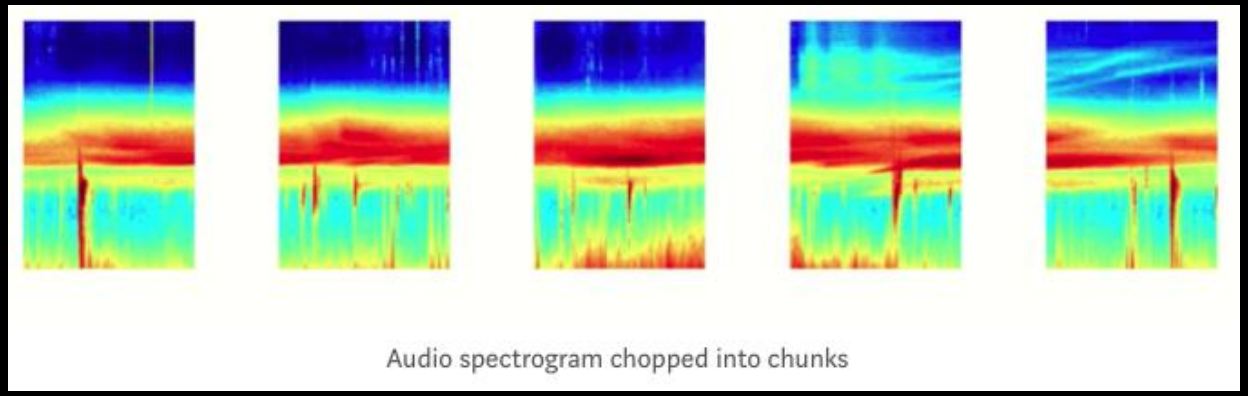
- Snapshots of a ball moving in time:



- You want to predict the direction it is moving
  - Now with the data you have about previous positions, you can predict more accurately

Sequence Data

- Audio:



Audio spectrogram chopped into chunks

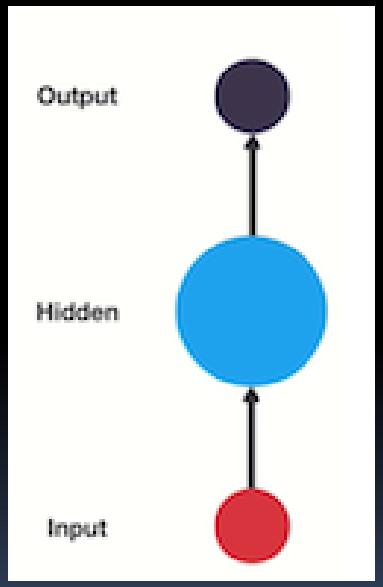
- Text Messaging:
  - I want to say
    hi that I

Sequential Memory

- Try saying the alphabet forward
- Now try saying it backwards
- Now say it forward, but start at the letter F
- Sequential memory makes it easier for your brain to recognize sequence patterns

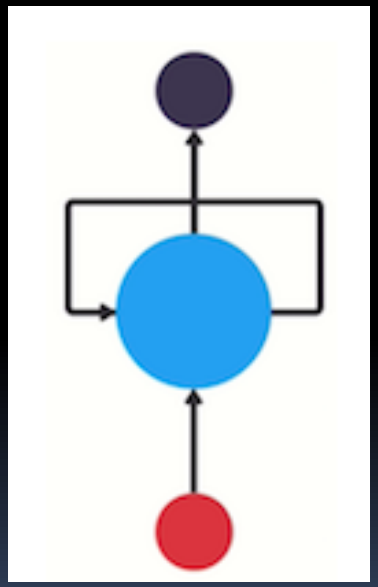
Recurrent Neural Networks

- Feed Forward Neural Network



Input information never touches a node twice

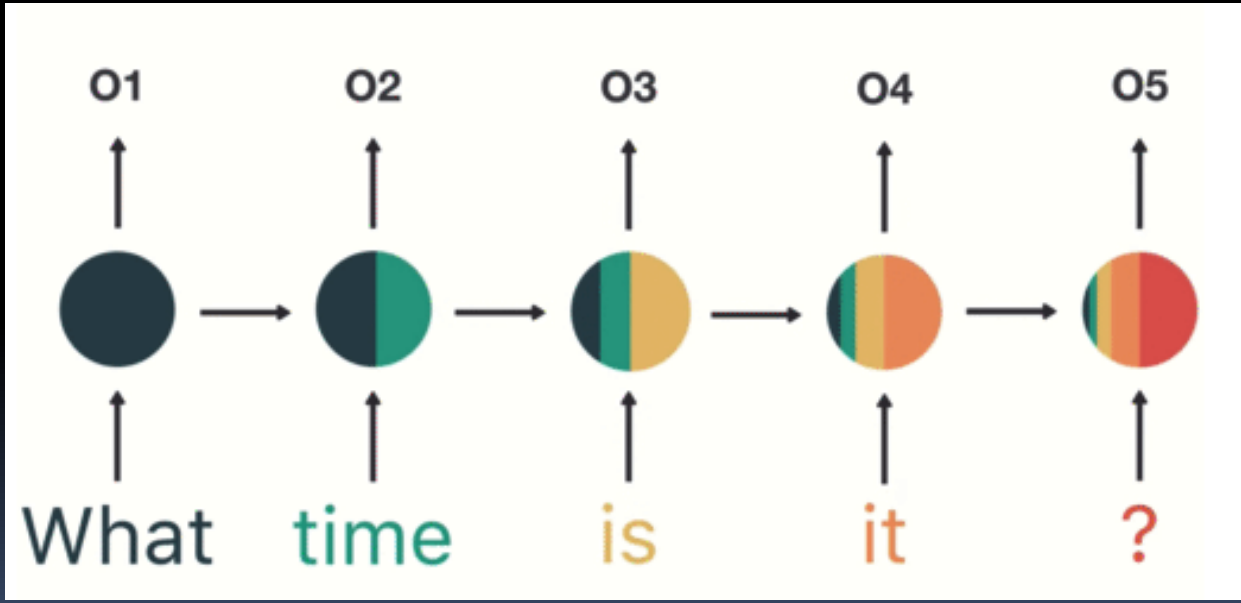
- Recurrent Neural Network



Input information cycles through a loop

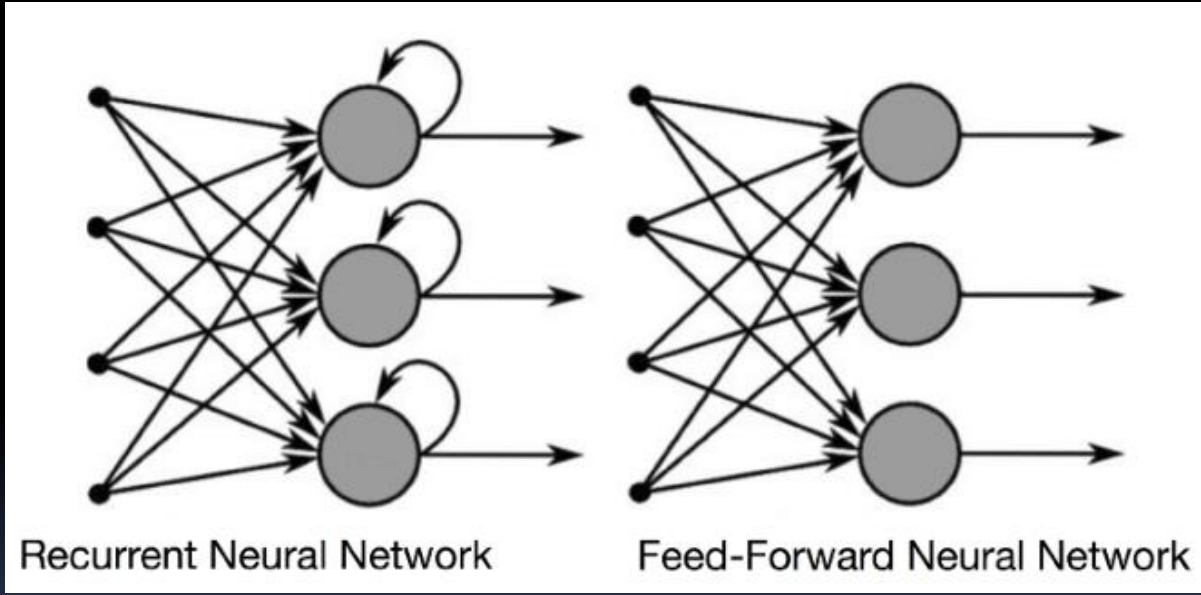
Recurrent Neural Networks

- Hidden state is retained and used as input in subsequent iterations



Recurrent Neural Networks

- Another view



Language Models

- Word ordering:
  - the cat is small vs. small is the cat
- Word choice:
  - walking home after school vs. walking house after school
- An incorrect but necessary Markov assumption:

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \qquad p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

Recurrent Neural Networks

- Input vector x_i (data)
- Output vector y_i (data)
- Predicted output vector $\hat{y}_i$ (computed through forward propagation)
- Hidden state h_i

Thus, we have three separate matrices of weights:

- Input-to-hidden weights W_{hx}
- Hidden-to-hidden weights W_{hh}
- Hidden-to-output weights W_{yh}

Recurrent Neural Networks

- Forward propagation:

$$h_i = \sigma(W_{hh} h_{i-1} + W_{hx} x_i + b_h)$$
$$\hat{y}_i = W_{yh} h_i$$

Recurrent Neural Networks

- Use same weights at each time step
- Condition network on all previous inputs
- RAM requirement scales with number of words, not number of combinations of words (n-grams)

$$x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$$

$$\begin{aligned} h_t &= \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right) \\ \hat{y}_t &= \text{softmax}\left(W^{(S)} h_t\right) \end{aligned}$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$

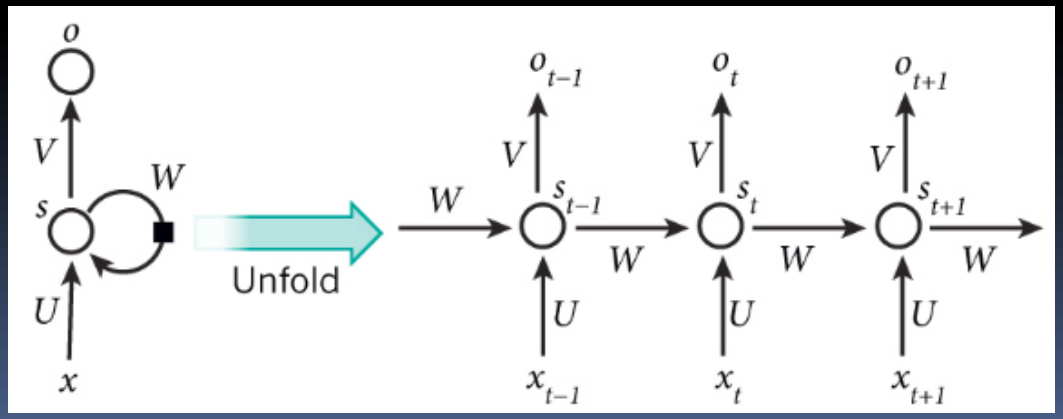
Recurrent Neural Networks

For each training epoch, we start by training on shorter sequences, and then train on progressively longer sequences. If our max sequence length is N, then we will first train on all 1-length sequences (the first element only), then all 2-element sequences (the first element followed by the second element), and so on, until we are training on N-length sequences.

For each length of sequence k, we unfold the network into a normal feedforward network that has k hidden layers. However, unlike a normal hidden layer, each hidden layer also takes an input - the input into the neural network at that time step. Thus, the network actually has $k + 1$ different inputs: an initial hidden state and k inputs, one per time step.

Back Propagation Through Time (BPTT)

- Back propagation on an unrolled recurrent neural network
  - Unrolling is a conceptual tool
  - View the RNN as a sequence of ANNs that you train one after the other



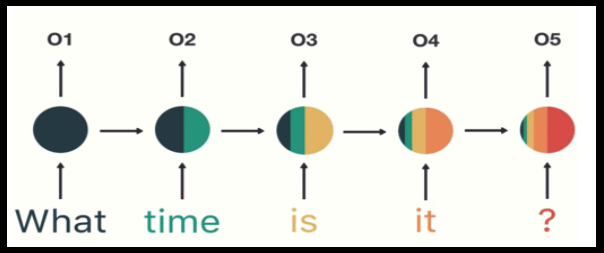
Backpropagation Through Time:

Initialization: *Initialize weight matrices* W_{hx} *(input-to-hidden weights),* W_{hh} *(hidden to hidden weights), and* W_{yh} *to random values.*

Repeat until convergence:

1. *Let* i *be the current sequence length, starting at 1.*
2. *Unfold your neural network in time for* i *time steps to create a standard feed-forward net.*
3. *Set the inputs to your feed-forward net to be the initial hidden state (a vector of zeros) and the input at every time step. Note that your network will have inputs at every hidden layer as well as at the initial input layer.*
4. *Perform forward and backward propagation as in a normal network.*
5. *Average all the gradients at each layer, so that the weight update to the matrices is the same in each time step.*
6. *Repeat steps 1 through 5 for* $i = 1$ *through* $i = N$*, where* N *is the maximum number of elements in your sequence data set.*

Vanishing Gradient



- AKA Short Term Memory
  - Due to the nature of back propagation
    - If the adjustments to a layer before the current one is small, the adjustments to the current layer will be smaller
    - Gradient shrinks exponentially
  - Back propagation through time (BPTT)
    - Gradient shrinks exponentially through each time step

LSTMs and GRUs

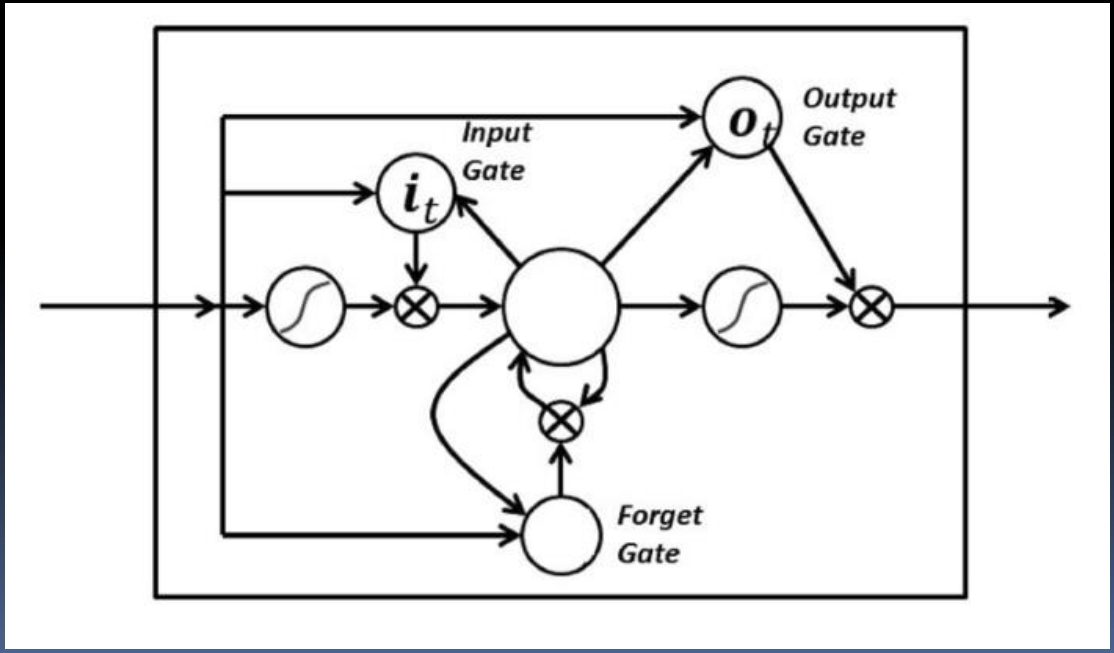
- LSTM – Long Short-Term Memory
  - Information is retained in memory
    - LSTM can read, write and delete this memory
- GRU – Gated Recurrent Units
  - Gates decide whether to store or delete information
    - Based on importance assigned
    - Assigning importance based on weights
- Both can learn what information to add or remove in a hidden state

LSTMs and GRUs

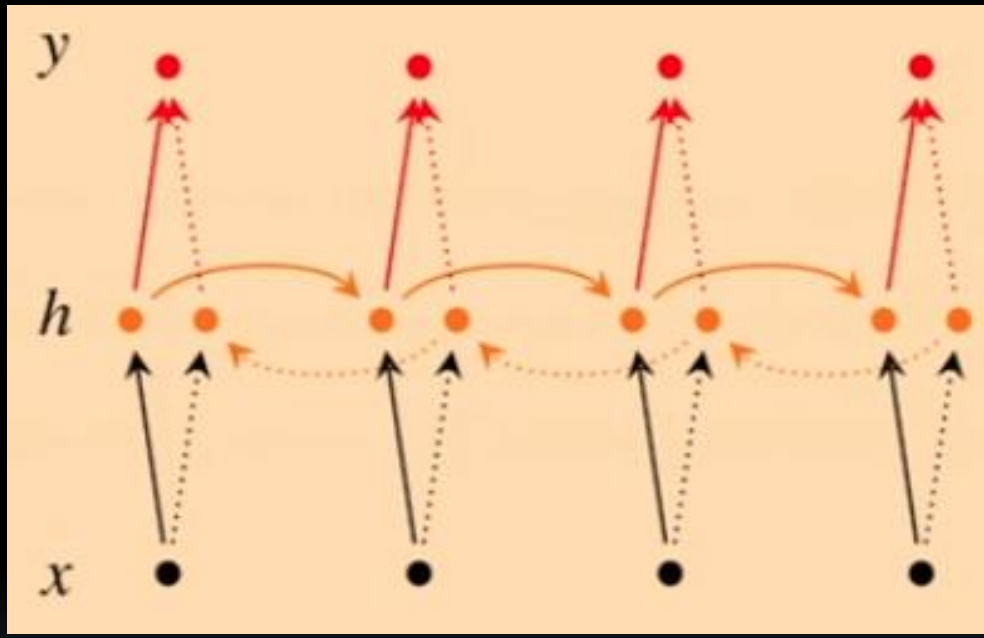
- Three gates:
  - Input
    - Let new input in
  - Forget
    - Delete information that isn’t important
  - Output
    - Let information impact current output

LSTMs and GRUs

- Gates are analog – often sigmoid – ranging from 0 to 1
- Can back propagate with them



Bidirectional RNNs

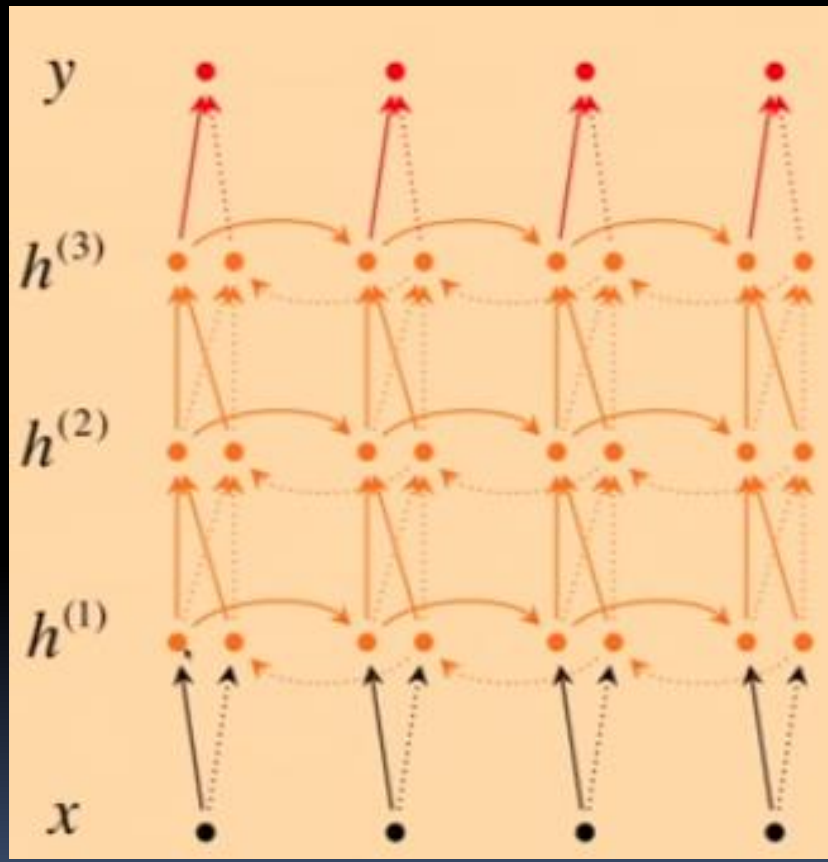


$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$y_t = g(U[\vec{h}_t; \overleftarrow{h}_t] + c)$$

Deep Bidirectional RNNs



$$\vec{h}_t^{(i)} = f(\vec{W}^{(i)} h_t^{(i-1)} + \vec{V}^{(i)} \vec{h}_{t-1}^{(i)} + \vec{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\vec{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Summary

- Introduction
- Sequence Data
- Sequential Memory
- Recurrent Neural Networks
- Vanishing Gradient
- LSTMs and GRUs