Hidden Markov Models

- Markov chains

\[
P(X_1) \quad P(X|X_{-1})
\]

- Hidden Markov models (HMMs)
  - Underlying Markov chain over states $S$
  - You observe outputs (effects) at each time step
An HMM is defined by:

- Initial distribution: $P(X_1)$
- Transitions: $P(X_t | X_{t-1})$
- Emissions: $P(E | X)$
Hidden Markov Models

- Defines a joint probability distribution:

\[
P(X_1, E_1, X_2, E_2, X_3, E_3) = P(X_1)P(E_1|X_1)P(X_2|X_1)P(E_2|X_2)P(X_3|X_2)P(E_3|X_3)
\]

\[
P(X_1, \ldots, X_n, E_1, \ldots, E_n) = P(X_1:n, E_1:n) = P(X_1)P(E_1|X_1) \prod_{t=2}^{N} P(X_t|X_{t-1})P(E_t|X_t)
\]

- Questions to be resolved:
  - Does this indeed define a joint distribution?
  - Can every joint distribution be factored this way, or are we making some assumptions about the joint distribution by using this factorization?
- From the chain rule, every joint distribution over $X_1, E_1, X_2, E_2, X_3, E_3$ can be written as:

$$P(X_1, E_1, X_2, E_2, X_3, E_3) = P(X_1)P(E_1|X_1)P(X_2|X_1, E_1)P(E_2|X_1, E_1, X_2)$$
$$P(X_3|X_1, E_1, X_2, E_2)P(E_3|X_1, E_1, X_2, E_2, X_3)$$

- Assuming that

$$X_2 \perp E_1 \mid X_1, \quad E_2 \perp X_1, E_1 \mid X_2, \quad X_3 \perp X_1, E_1, E_2 \mid X_2, \quad E_3 \perp X_1, E_1, X_2, E_2 \mid X_3$$

gives us the expression posited on the previous slide:

$$P(X_1, E_1, X_2, E_2, X_3, E_3) = P(X_1)P(E_1|X_1)P(X_2|X_1)P(E_2|X_2)P(X_3|X_2)P(E_3|X_3)$$
Chain Rule and HMMs

- From the chain rule, every joint distribution over \( X_1, E_1, \ldots, X_T, E_T \) can be written as:

\[
P(X_1, E_1, \ldots, X_T, E_T) = P(X_1)P(E_1|X_1) \prod_{t=2}^{T} P(X_t|X_{t-1})P(E_t|X_{t-1}, X_t)
\]

- Assuming that for all \( t \):
  - State independent of all past states and all past evidence given the previous state, i.e.:
    \[
    X_t \perp X_1, E_1, \ldots, X_{t-2}, E_{t-2}, E_{t-1} \mid X_{t-1}
    \]
  - Evidence is independent of all past states and all past evidence given the current state, i.e.:
    \[
    E_t \perp X_1, E_1, \ldots, X_{t-2}, E_{t-2}, X_{t-1}, E_{t-1} \mid X_t
    \]

  gives us the expression posited on the earlier slide:

\[
P(X_1, E_1, \ldots, X_T, E_T) = P(X_1)P(E_1|X_1) \prod_{t=2}^{T} P(X_t|X_{t-1})P(E_t|X_t)
\]
Real HMM Examples

- Speech recognition HMMs:
  - Observations are acoustic signals (continuous valued)
  - States are specific positions in specific words (so, tens of thousands)
Real HMM Examples

- **Robot tracking:**
  - Observations are range readings (continuous)
  - States are positions on a map (continuous)
Example: Pairs of Sequences

- Consider the problem of jointly modeling a pair of strings
  - E.g.: part of speech tagging

<table>
<thead>
<tr>
<th>DT</th>
<th>NNP</th>
<th>NN</th>
<th>VBD</th>
<th>VBN</th>
<th>RP</th>
<th>NN</th>
<th>NNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT</td>
<td>NN</td>
<td>IN</td>
<td>NN</td>
<td>VBD</td>
<td>NNS</td>
<td>VBD</td>
<td></td>
</tr>
</tbody>
</table>

The Georgia branch had taken on loan commitments ...

The average of interbank offered rates plummeted ...

- Q: How do we map each word in the input sentence onto the appropriate label?
- A: We can learn a joint distribution:

\[ p(x_1 \ldots x_n, y_1 \ldots y_n) \]

- And then compute the most likely assignment:

\[ \arg \max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n) \]
Classic Solution: HMMs

- We want a model of sequences $y$ and observations $x$

\[
p(x_1 \ldots x_n, y_1 \ldots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1})e(x_i|y_i)
\]

where $y_0=\text{START}$ and we call $q(y'|y)$ the transition distribution and $e(x|y)$ the emission (or observation) distribution.

- Assumptions:
  - Tag/state sequence is generated by a markov model
  - Words are chosen independently, conditioned only on the tag/state
Example: POS Tagging

The Georgia branch had taken on loan commitments …
HMM Inference and Learning

- **Learning**
  - Maximum likelihood: transitions $q$ and emissions $e$

\[
p(x_1 \ldots x_n, y_1 \ldots y_n) = q(\text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1})e(x_i|y_i)
\]

- **Inference (linear time in sentence length!)**
  - Viterbi: $y^* = \arg \max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)$
Learning: Maximum Likelihood

\[ p(x_1 \ldots x_n, y_1 \ldots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i) \]

- **Learning**
  - Maximum likelihood methods for estimating transitions \( q \) and emissions \( e \)

\[ q_{ML}(y_i|y_{i-1}) = \frac{c(y_{i-1}, y_i)}{c(y_{i-1})} \quad e_{ML}(x|y) = \frac{c(y, x)}{c(y)} \]

- Will these estimates be high quality?
  - Which is likely to be more sparse, \( q \) or \( e \)?
- Can use smoothing tricks
Learning: Low Frequency Words

\[ p(x_1 \ldots x_n, y_1 \ldots y_n) = q(\text{STOP} | y_n) \prod_{i=1}^{n} q(y_i | y_{i-1}) e(x_i | y_i) \]

- Typically, linear interpolation works well for transitions
  \[ q(y_i | y_{i-1}) = \lambda_1 q_{ML}(y_i | y_{i-1}) + \lambda_2 q_{ML}(y_i) \]

- However, other approaches used for emissions
  - **Step 1:** Split the vocabulary
    - *Frequent words:* appear more than M (often 5) times
    - *Low frequency:* everything else
  - **Step 2:** Map each low frequency word to one of a small, finite set of possibilities
    - For example, based on prefixes, suffixes, etc.
  - **Step 3:** Learn model for this new space of possible word sequences
Low Frequency Words: An Example

**Named Entity Recognition** [Bickel et. al, 1999]
- Used the following word classes for infrequent words:

<table>
<thead>
<tr>
<th>Word class</th>
<th>Example</th>
<th>Intuition</th>
</tr>
</thead>
<tbody>
<tr>
<td>twoDigitNum</td>
<td>90</td>
<td>Two digit year</td>
</tr>
<tr>
<td>fourDigitNum</td>
<td>1990</td>
<td>Four digit year</td>
</tr>
<tr>
<td>containsDigitAndAlpha</td>
<td>A8956-67</td>
<td>Product code</td>
</tr>
<tr>
<td>containsDigitAndDash</td>
<td>09-96</td>
<td>Date</td>
</tr>
<tr>
<td>containsDigitAndSlash</td>
<td>11/9/89</td>
<td>Date</td>
</tr>
<tr>
<td>containsDigitAndComma</td>
<td>23,000.00</td>
<td>Monetary amount</td>
</tr>
<tr>
<td>containsDigitAndPeriod</td>
<td>1.00</td>
<td>Monetary amount, percentage</td>
</tr>
<tr>
<td>othernum</td>
<td>456789</td>
<td>Other number</td>
</tr>
<tr>
<td>allCaps</td>
<td>BBN</td>
<td>Organization</td>
</tr>
<tr>
<td>capPeriod</td>
<td>M.</td>
<td>Person name initial</td>
</tr>
<tr>
<td>firstWord</td>
<td>first word of sentence</td>
<td>no useful capitalization information</td>
</tr>
<tr>
<td>initCap</td>
<td>Sally</td>
<td>Capitalized word</td>
</tr>
<tr>
<td>lowercase</td>
<td>can</td>
<td>Uncapitalized word</td>
</tr>
<tr>
<td>other</td>
<td>,</td>
<td>Punctuation marks, all other words</td>
</tr>
</tbody>
</table>
Inference (Decoding)

- Problem: find the most likely (Viterbi) sequence under the model

\[
\arg \max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)
\]

- Given model parameters, we can score any sequence pair

\[
\text{Fed} \quad \text{raises} \quad \text{interest} \quad \text{rates} \quad 0.5 \quad \text{percent} .
\]

\[
q(\text{NNP}|\bullet) \cdot e(\text{Fed}|\text{NNP}) \cdot q(\text{VBZ}|\text{NNP}) \cdot e(\text{raises}|\text{VBZ}) \cdot q(\text{NN}|\text{VBZ}) .
\]

- In principle, we’re done – list all possible tag sequences, score each one, pick the best one (the Viterbi state sequence)

\[
\text{NNP} \quad \text{VBZ} \quad \text{NN} \quad \text{NNS} \quad \text{CD} \quad \text{NN} \quad \rightarrow \quad \log P = -23
\]

\[
\text{NNP} \quad \text{NNS} \quad \text{NN} \quad \text{NNS} \quad \text{CD} \quad \text{NN} \quad \rightarrow \quad \log P = -29
\]

\[
\text{NNP} \quad \text{VBZ} \quad \text{VB} \quad \text{NNS} \quad \text{CD} \quad \text{NN} \quad \rightarrow \quad \log P = -27
\]
**Finding the Best Trajectory**

- Too many trajectories (state sequences) to list
- **Option 1: Beam Search**

  - A beam is a set of partial hypotheses
  - Start with just the single empty trajectory
  - At each derivation step:
    - Consider all continuations of previous hypotheses
    - Discard most, keep top \( k \)

  - **Beam search works ok in practice**
    - … but sometimes you want the optimal answer
    - … and there’s usually a better option than naïve beams
HMM Computations: Inference

- **Given**
  - joint $P(X_{1:n}, E_{1:n})$
  - evidence $E_{1:n} = e_{1:n}$

- **Inference problems include:**
  - **Filtering**, find $P(X_t|e_{1:t})$ for current $t$
  - **Smoothing**, find $P(X_t|e_{1:n})$ for past $t$
  - **Most probable explanation**, find
    $$x_{1:n}^* = \arg\max_{x_{1:n}} P(x_{1:n}|e_{1:n})$$
Inference Recap: Simple Cases

\[ P(X_1) \]

\[ P(E|X) \]

\[ P(X_1|e_1) \]

\[ P(x_1|e_1) = \frac{P(x_1, e_1)}{P(e_1)} \]

\[ \propto_{X_1} P(x_1, e_1) \]

\[ = P(x_1)P(e_1|x_1) \]

\[ P(X_t|X_{t-1}) \]

\[ X_1 \rightarrow X_2 \]

\[ P(x_2) = \sum_{x_1} P(x_1, x_2) \]

\[ = \sum_{x_1} P(x_1)P(x_2|x_1) \]
Passage of Time

- We want to know: \( B_t(X) = P(X_t|e_{1:t}) \)
- We can derive the following updates

\[
P(X_{t+1}|e_{1:t}) = \sum_{x_t} P(X_{t+1}, x_t|e_{1:t})
\]

\[
= \sum_{x_t} P(X_{t+1}|x_t, e_{1:t}) P(x_t|e_{1:t})
\]

\[
= \sum_{x_t} P(X_{t+1}|x_t) P(x_t|e_{1:t})
\]

- To get \( B_t(X) \) compute each entry and normalize
Assume we have current belief $P(X | \text{previous evidence})$:

\[
P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}, e_{t+1}|e_{1:t}) / P(e_{t+1}|e_{1:t})
\]
\[
\propto P(X_{t+1}, e_{t+1}|e_{1:t})
\]
\[
= P(e_{t+1}|e_{1:t}, X_{t+1}) P(X_{t+1}|e_{1:t})
\]
\[
= P(e_{t+1}|X_{t+1}) P(X_{t+1}|e_{1:t})
\]
Online Belief Updates

- Every time step, we start with current $P(X \mid \text{evidence})$
- We update for time:

\[
P(x_t|e_{1:t-1}) = \sum_{x_{t-1}} P(x_{t-1}|e_{1:t-1}) \cdot P(x_t|x_{t-1})
\]

- We update for evidence:

\[
P(x_t|e_{1:t}) \propto_x P(x_t|e_{1:t-1}) \cdot P(e_t|x_t)
\]
The Forward Algorithm

- We want to know:  \( B_t(X) = P(X_t|e_{1:t}) \)
- We can derive the following updates

\[
P(x_t|e_{1:t}) \propto_X P(x_t, e_{1:t})
\]

\[
= \sum_{x_{t-1}} P(x_{t-1}, x_t, e_{1:t})
\]

\[
= \sum_{x_{t-1}} P(x_{t-1}, e_{1:t-1}) P(x_t|x_{t-1}) P(e_t|x_t)
\]

\[
= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}, e_{1:t-1})
\]

- To get \( B_t(X) \) compute each entry and normalize
- Problem: space is \(|X|\) and time is \(|X|^2\) per time step
The State Lattice / Trellis

The diagram represents a state lattice with transitions labeled by events. The events are:
- Fed
- raises
- interest
- rates
- STOP

The transitions are labeled with probabilities and are directed from one state to another. The states are represented by circles, and the transitions are marked with arrows indicating the events and their probabilities.

For example, the transition from the state labeled 'Fed' to the state labeled 'raises' is labeled with the event 'e(Fed|N)' and the transition from 'raises' to 'interest' is labeled with 'e(raises|V)' and so on.

This lattice is used to model the sequence of events in a specific context, such as economic policies and their impacts.
Dynamic Programming!

\[
\arg \max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)
\]

- Define \( \pi(i, y_i) \) to be the max score of a sequence of length \( i \) ending in tag \( y_i \)

\[
\pi(i, y_i) = \max_{y_1 \ldots y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_i)
\]

\[
= \max_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \max_{y_1 \ldots y_{i-2}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-1})
\]

\[
= \max_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \pi(i - 1, y_{i-1})
\]

- We now have an efficient algorithm. Start with \( i=0 \) and work your way to the end of the sentence!
The Viterbi Algorithm

- **Dynamic program for computing (for all i)**

  \[ \pi(i, y_i) = \max_{y_1 \ldots y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_i) \]

- **Iterative computation**

  \[ \pi(0, y_0) = \begin{cases} 
  1 & \text{if } y_0 == \text{START} \\
  0 & \text{otherwise} 
  \end{cases} \]

  For \( i = 1 \ldots n \):

  \[ \pi(i, y_i) = \max_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \pi(i - 1, y_{i-1}) \]

- **Also, store back pointers**

  \[ bp(i, y_i) = \arg \max_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \pi(i - 1, y_{i-1}) \]
<table>
<thead>
<tr>
<th></th>
<th>Fruit Flies</th>
<th>Like</th>
<th>Bananas</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\pi(1, N)$</td>
<td>$\pi(2, N)$</td>
<td>$\pi(3, N)$</td>
</tr>
<tr>
<td></td>
<td>$\pi(1, V)$</td>
<td>$\pi(2, V)$</td>
<td>$\pi(3, V)$</td>
</tr>
<tr>
<td></td>
<td>$\pi(1, IN)$</td>
<td>$\pi(2, IN)$</td>
<td>$\pi(3, IN)$</td>
</tr>
</tbody>
</table>

Fruit Flies Like Bananas

START

STOP

Fruit Flies Like Bananas
Fruit Flies Like Bananas

\[
\begin{align*}
\pi(1, N) &= 0.03 \\
\pi(1, V) &= 0.01 \\
\pi(1, IN) &= 0
\end{align*}
\]
Fruit Flies Like Bananas

\[ \pi(1, N) = 0.03 \]
\[ \pi(1, V) = 0.01 \]
\[ \pi(1, IN) = 0 \]

\[ \pi(2, N) = 0.005 \]
\[ \pi(2, V) \]
\[ \pi(2, IN) \]

\[ \pi(3, N) \]
\[ \pi(3, V) \]
\[ \pi(3, IN) \]

\[ \pi(4, N) \]
\[ \pi(4, V) \]
\[ \pi(4, IN) \]
Fruit Flies Like Bananas

\[
\begin{align*}
\pi(1, N) &= 0.03 \\
\pi(1, V) &= 0.01 \\
\pi(1, IN) &= 0 \\
\pi(2, N) &= 0.005 \\
\pi(2, V) &= 0.007 \\
\pi(2, IN) &= 0 \\
\pi(3, N) \\
\pi(3, V) \\
\pi(3, IN) \\
\pi(4, N) \\
\pi(4, V) \\
\pi(4, IN)
\end{align*}
\]
Fruit Flies Like Bananas

\[ \pi(1, N) = 0.03 \]
\[ \pi(2, N) = 0.005 \]
\[ \pi(3, N) = 0.0001 \]

\[ \pi(1, V) = 0.01 \]
\[ \pi(2, V) = 0.007 \]
\[ \pi(3, V) = 0.0007 \]

\[ \pi(1, IN) = 0 \]
\[ \pi(2, IN) = 0 \]
\[ \pi(3, IN) = 0.0003 \]
Fruit Flies Like Bananas

\[ \pi(1, N) = 0.03 \]
\[ \pi(2, N) = 0.005 \]
\[ \pi(3, N) = 0.0001 \]
\[ \pi(4, N) = 0.00003 \]

\[ \pi(1, V) = 0.01 \]
\[ \pi(2, V) = 0.007 \]
\[ \pi(3, V) = 0.0007 \]
\[ \pi(4, V) = 0.00001 \]

\[ \pi(1, IN) = 0 \]
\[ \pi(2, IN) = 0 \]
\[ \pi(3, IN) = 0.0003 \]
\[ \pi(4, IN) = 0 \]
Fruit Flies Like Bananas

\[
\begin{align*}
\pi(1, N) &= 0.03 \\
\pi(1, V) &= 0.01 \\
\pi(1, IN) &= 0 \\
\pi(2, N) &= 0.005 \\
\pi(2, V) &= 0.007 \\
\pi(2, IN) &= 0 \\
\pi(3, N) &= 0.0001 \\
\pi(3, V) &= 0.0007 \\
\pi(3, IN) &= 0.0003 \\
\pi(4, N) &= 0.00003 \\
\pi(4, V) &= 0.00001 \\
\pi(4, IN) &= 0
\end{align*}
\]
Fruit Flies Like Bananas

\[
\begin{align*}
\pi(1, N) &= 0.03 \\
\pi(1, V) &= 0.01 \\
\pi(1, IN) &= 0 \\
\pi(2, N) &= 0.005 \\
\pi(2, V) &= 0.007 \\
\pi(2, IN) &= 0 \\
\pi(3, N) &= 0.0001 \\
\pi(3, V) &= 0.0007 \\
\pi(3, IN) &= 0.0003 \\
\pi(4, N) &= 0.00003 \\
\pi(4, V) &= 0.00001 \\
\pi(4, IN) &= 0.00003 \\
\end{align*}
\]
The Viterbi Algorithm: Runtime

- Linear in sentence length $n$
- Polynomial in the number of possible tags $|K|$

\[ \pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i - 1, y_{i-1}) \]

- Specifically:

  \[ O(n|K|) \] entries in $\pi(i, y_i)$

  \[ O(|K|) \] time to compute each $\pi(i, y_i)$

- Total runtime: \[ O(n|K|^2) \]

- Q: Is this a practical algorithm?
- A: depends on $|K|$....
- Recurrent Neural Models
Vanilla Neural Network

one to one

Vanilla Neural Networks
Recurrent Neural Networks: Process Sequences

- **One to one**: Image captioning
- **One to many**: Sentiment Classification
- **Many to one**: Machine Translation
- **Many to many**: Video Classification
Recurrent Neural Networks

usually want to predict a vector at some time steps
Recurrent Neural Networks

We can process a sequence of vectors $\mathbf{x}$ by applying a **recurrence formula** at every time step:

$$ h_t = f_W(h_{t-1}, x_t) $$

new state / old state / input vector at some time step

some function with parameters $W$

Notice: the same function with the same set of parameters are used at each time step.
(Vanilla) Recurrent Neural Network

The state consists of a single "hidden" vector $h$:

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$
RNN: computational graph
RNN: Computational Graph: Many to One
RNN: Computational Graph: Many to Many
Example:
Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”
Example:
Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
Vanilla Gradient Flow

Backpropagation from $h_t$ to $h_{t-1}$ multiplies by $W$ (actually $W_{hh}^T$).

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$= \tanh \left( (W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

$$= \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$
Vanilla Gradient Flow

Computing gradient of $h_0$ involves many factors of $W$ (and repeated tanh)

Gradients might become really small
Vanishing gradients
RNN Variants

Long Short Term Memory (LSTM)

Vanilla RNN

\[ h_t = \tanh\left( W \left( h_{t-1}, x_t \right) \right) \]

LSTM

\[
\begin{pmatrix}
    i \\
    f \\
    o \\
    g
\end{pmatrix}
= \begin{pmatrix}
    \sigma \\
    \sigma \\
    \sigma \\
    \tanh
\end{pmatrix}
W \begin{pmatrix}
    h_{t-1} \\
    x_t
\end{pmatrix}
\]

\[ c_t = f \odot c_{t-1} + i \odot g \]
\[ h_t = o \odot \tanh(c_t) \]
Long Short Term Memory (LSTM)  
[Hochreiter et al., 1997]

- **f**: Forget gate, Whether to erase cell
- **i**: Input gate, whether to write to cell
- **g**: Gate gate (?), How much to write to cell
- **o**: Output gate, How much to reveal cell

\[
\begin{pmatrix}
i \\
f \\
o \\
g
\end{pmatrix} =
\begin{pmatrix}
\sigma \\
\sigma \\
\sigma \\
tanh
\end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}
\]

\[
c_t = f \odot c_{t-1} + i \odot g
\]

\[
h_t = o \odot \tanh(c_t)
\]
Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

\[
\begin{align*}
\sigma & = \text{sigmoid} \\
\tanh & = \text{tanh} \\
\end{align*}
\]

\[
\begin{pmatrix}
i \\ f \\ o \\ g \\
\end{pmatrix} =
\begin{pmatrix}
\sigma \\ \sigma \\ \sigma \\ \tanh
\end{pmatrix}
W \begin{pmatrix}
h_{t-1} \\ x_t
\end{pmatrix}
\]

\[
c_t = f \odot c_{t-1} + i \odot g
\]

\[
h_t = o \odot \text{tanh}(c_t)
\]
Summary

- HMMs are used for modeling sequential data
  - Learning
  - Inference
- RNNs are neural architecture to model sequential data
- Vanilla RNNs are simple, but don’t work in practice
- Other variants such as LSTMs are being used