CPSC 340:
Machine Learning and Data Mining

Markov Chains

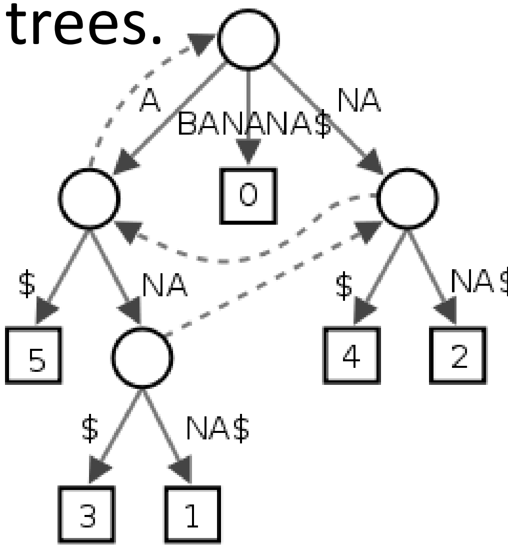
Fall 2015

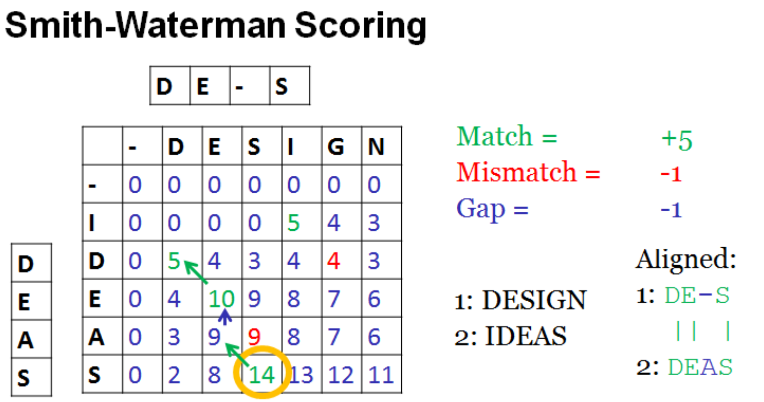
Admin

- Assignment 6 due Friday.
- Final exam details:
  - December 15: 8:30-11 (WESB 100).
  - 4 pages of cheat sheet allowed.
  - 9 questions.
  - Practice questions and list of topics posted.

Last Time: Sequence Alignment

- We discussed finding similar sequences or aligning sequence parts.
  - Can find longest common substrings in linear time using suffix trees.
  - Using dynamic programming, we can compute ‘edit’ distance:
    - How many insertions/deletions/replacements to transform A into B?
  - Local alignment:
    - Find local regions with small edit distance.
  - BLAST:
    - Fast substring search to prune search for local alignments.
  - Multiple sequence alignment:
    - Hierarchical clustering helps align multiple sequences.





https://en.wikipedia.org/wiki/Suffix_tree
http://2012.igem.org/Team:Johns_Hopkins-Software/Cloud

Last Time: Dynamic Programming

- Dynamic programming:
  - Solves seemingly exponential-sized problems in polynomial-time.
- 3 ingredients:
  1. Given results of recursive calls, can solve problem efficiently.
  2. Limited number of possible arguments recursive calls.
  3. Memorize the results of recursive calls.
- Standard ways to implement dynamic programming:
  1. Start from final result, use recursion but ‘check’ if result is in global table.
  2. Bottom-up: start filling out entries in the table in order.

Example: Matrix Chain Multiplication

- Supposed we want to multiply matrices of different sizes:
  - ABCDE.
- Cost depends on order of multiplication:
  - A(B(C(DE))) vs. A((BC)(DE)).

A: 10 x 10000
B: 10000 x 10
C: 10 x 10000
D: 10000 x 10
E: 10 x 1

A(B(C(DE)))
10000 x 1
10 x 1
10000 x 1
10 x 1
Cheap!

A((BC)(DE))
10000 x 10000
10000 x 1
10000 x 1
10 x 1
expensive

- What is the optimal order?
  - There are an exponential number of possible orders.
  - With ‘n’ matrices, we can solve this in $O(n^3)$ using dynamic programming.

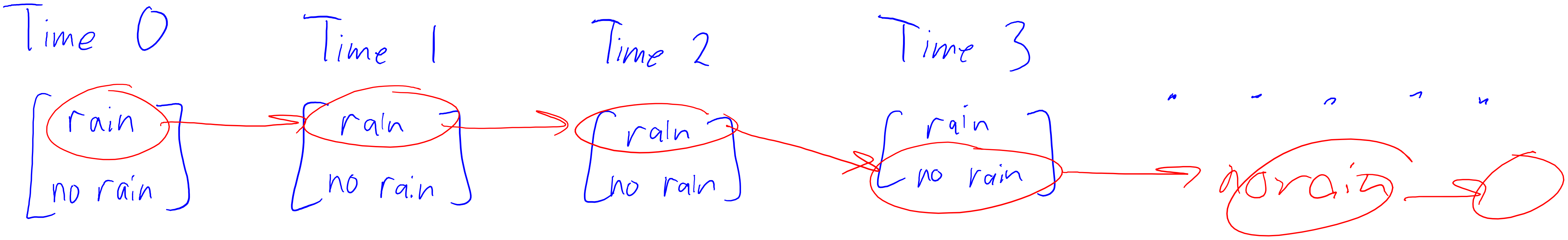
Example: Matrix Chain Multiplication

- Define the solution recursively:
  - Cost(ABCDE) is minimum of:
    - Cost(ABCD) plus Cost(E) plus cost of combining result.
    - Cost(ABC) plus Cost(~~C~~DE) plus cost of combining result.
    - Cost(AB) plus Cost(CDE) plus cost of combining result.
    - Cost(A) plus Cost(BCDE) plus cost of combining result.
  - We could solve problem in O(n) given recursive calls.
- There are only $O(n^2)$ possible recursive calls:
  - Ordering restricts possible recursions:
    - Cost(A), Cost(B), Cost(C), Cost(D), Cost(E),
    - Cost(AB), Cost(BC), Cost(CD), Cost(DE), Cost(ABC), Cost(BCD), Cost(CDE),
    - Cost(ABCD), Cost(BCDE).
- If you load results instead of re-computing, total cost is $O(n^3)$.

```
Cost(X1, X2, X3, ...)
{
  if (already computed)
  {
    load result
    return
  }
  else
  {
    do the work...
    store result
```

Markov Chain Example

- Markov chains have a set of ‘times’ and possible ‘states’:



Our 'states' are 'rain' and 'not rain', at each time you have to be in one of these states.

We define probabilities over initial state and transition between states.

$p(x_0 = \text{'rain'})$ is probability of it raining at time 0.

$p(x_t = \text{'rain'} \mid x_{t-1} = \text{'not rain'})$ is probability of it raining at time 't' if it was not raining at time t-1.

Markov Chains

- Modeling the probability of a sequence $x_0, x_1, x_2, x_3, \ldots$
  - At 'time' 0, we have a probability $p(x_0 = s)$ that 'x_0' will be in each 'state' 's'.
  - At 'time' t, we have probability $p_t(x_t = s_t \mid x_{t-1} = s_{t-1}, x_{t-2} = s_{t-2}, \ldots, x_0 = s_0)$:
    - Probability that 'x_t' is in state 's_t', given what has happened so far.
  - Based on product rule: $p(x_t, x_{t-1}, \ldots, x_0) = p(x_t \mid x_{t-1}, x_{t-2}, \ldots, x_0)\, p(x_{t-1}, x_{t-2}, \ldots, x_0)$

$$= \prod_{i=1}^{t} \left[p(x_i \mid x_{i-1}, x_{i-2}, \ldots, x_0) \right] p(x_0)$$

- Markov chains assume the Markov property:
  - Conditional independence assumption: $x_t \perp x_{t-2}, x_{t-1}, \ldots, x_0 \mid x_{t-1}$:
    - $p_t(x_t = s_t \mid x_{t-1} = s_{t-1}, x_{t-2} = s_{t-2}, \ldots, x_0 = s_0) = p_t(x_t = s_t \mid x_{t-1} = s_{t-1})$.
  - 'Memorylessness':

So $p(x_t, x_{t-1}, \ldots, x_0) = p(x_0) \prod_{i=1}^{t} p(x_i \mid x_{i-1})$

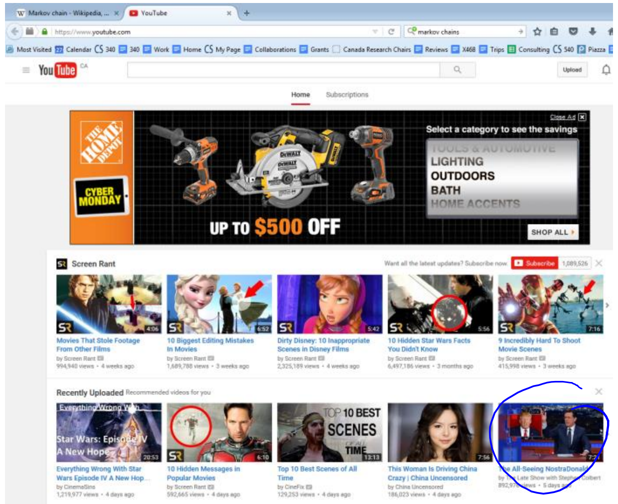
    - Probability only depends on where you are now, not where you were before.

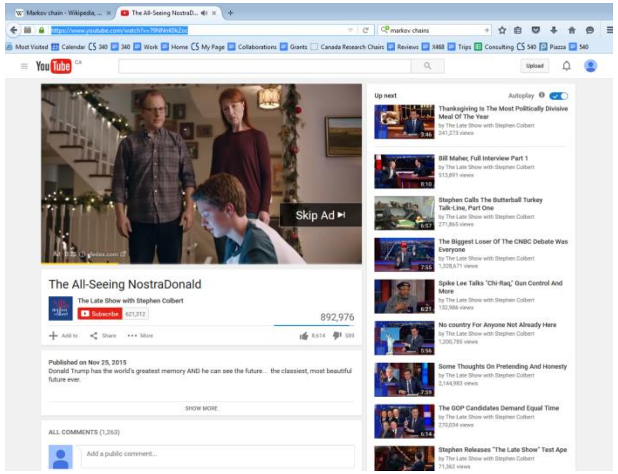
Markov Chain Examples

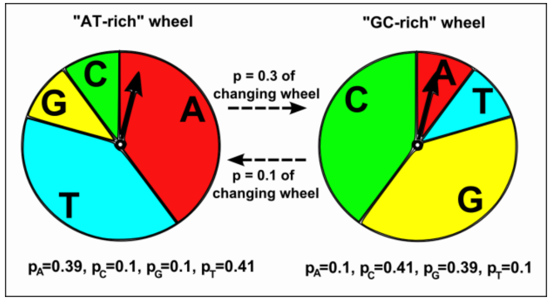
- We have already seen several examples:
  - PageRank, spectral clustering, graph-based SSL (A6Q2).
- PageRank example:
  - Each 'state' is a webpage on the internet.
  - We start on a random page: $p(x_0 = s) = 1/|S|$.
    - E.g., www.youtube.com (not really random).
  - At each time 't', we click on a random link:

$$p(x_t = s_2 \mid x_{t-1} = s_1) = \frac{1}{d_{s_1}} I[s_1 \text{ to } s_2 \text{ link exists}]$$

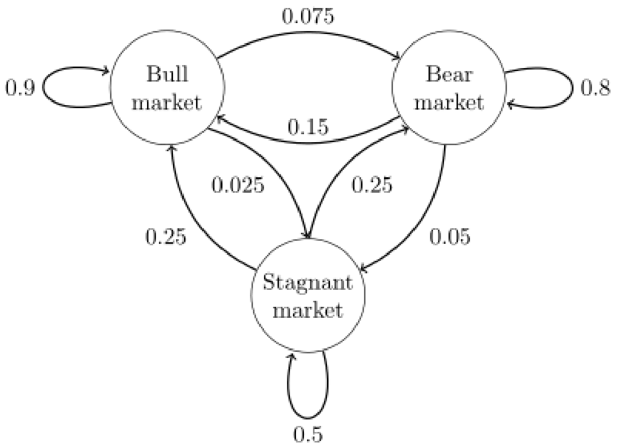
  - If allow moving to new random page ('damping'):
    - Markov property is still satisfied.
  - If time 't' depends on time 't-5':
    - Markov property is not satisfied (can't look at browser history).







Markov Chain Examples



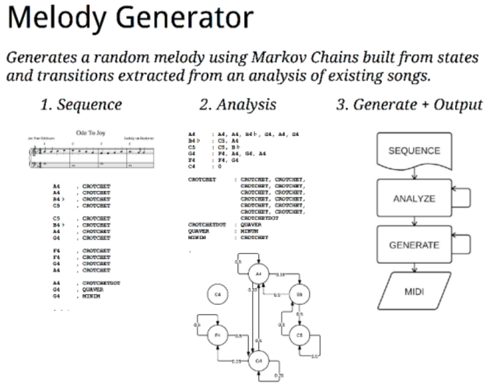
- In our examples, probabilities were homogeneous:

$$p(x_t = s_2 \mid x_{t-1} = s) = p(x_i = s_2 \mid x_{i-1} = s_1)$$

for any 'i' > 0

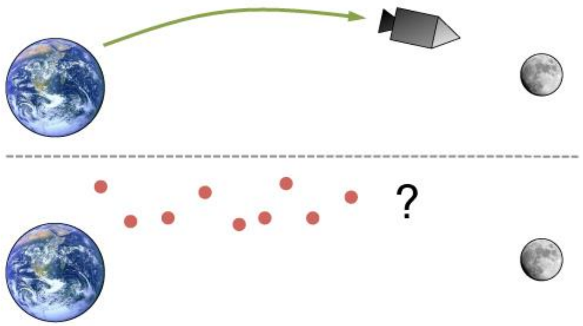
  – Important special case.

  – In practice, we often allow time-dependent probabilities.



- Incredible number of other applications:

  – Bioinformatics, physics/chemistry, speech recognition, predator-prey models, language tagging/generation, computing integrals, economic models, tracking missiles/players, modeling music.



https://en.wikipedia.org/wiki/Markov_chain
http://www.cs.uml.edu/ecg/index.php/AIfall11/MarkovMelodyGenerator
http://a-little-book-of-r-for-bioinformatics.readthedocs.org/en/latest/src/chapter10.html
https://plus.maths.org/content/understanding-unseen
http://www.cs.ubc.ca/~okumak/research.html

Markov Chain Tasks

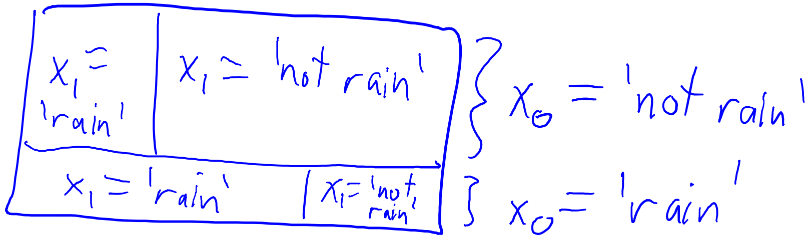
- We are going to focus on discrete time and states.
- Common tasks we want to do with Markov chains:
  1. Sampling: given model, simulate from $p(x_t, x_{t-1}, \dots, x_0)$.
  2. Learning: estimating $p(x_t = s_1 \mid x_t = s_2)$ to make model.
  3. Inference: given model, compute $p_t(x_t = s)$.
  4. Stationary distribution: $p_\infty(x_\infty = s)$.
  5. Decoding: $\max_{y1,y2,\dots,yt} p(x_t, x_{t-1}, x_{t-2}, \dots, x_0)$.
  6. Conditional inference: $p(x_t = s_1 \mid x_{t-1} = s_2, x_{t+10} = s_3)$.

Sampling from Markov Chains

- Sampling from a Markov chain:
  - Generate a sequence $x_0, x_1, \dots x_t$ following the joint distribution:

$$p(x_t, x_{t-1}, \dots, x_0) = p(x_0) \prod_{i=1}^{t} p(x_i \mid x_{i-1})$$

  - E.g., can we simulate a 'random web surfer'?
- Easy for discrete time/states, a random walk model:
  - Generate x_0 according to $p(x_0)$.
  - For i = 1,2,…,t
    - Given x_{i-1}, generate x_i according to $p(x_i \mid x_{i-1})$.
- Why this works: Probability space



Random walk gives each area the correct probability.

Learning Parameters of Markov Chain

- Learning in Markov chains:
  - Given sample(s) of Markov chain, estimate what probabilities should be.
- Maximum likelihood estimates:
  - $p(x_0 = s) = N(x_0 = s)/N$.

    → number of samples we have.

    → number of times we start with $x_0 = s$.
  - Inhomogeneous case:
    - $p_t(x_t = s_2 \mid x_{t-1} = s_1) = N(x_{t-1} = s_1, x_t = s_2)/N(x_{t-1} = s_1)$.

      → number of times we were in s_1 at time $(t-1)$.

      → number of samples where we went from s_1 to s_2 at time $(t-1)$
  - Homogeneous case:
    - $p(x_t = s_2 \mid x_{t-1} = s_1) = N(x_{i-1} = s_1, x_i = s_2)/N(x_{i-1} = s_1)$ for any 'i'.

      → number of times we were in s_1.

      → number of times we went from s_1 to s_2.
- Initial $p(x_0 = s)$ and inhomogeneous case need multiple sequences.
  - Need a lot of data if number of states is very large.

Inference in Markov Chains

- Inference: compute probability of being in state 's' at time 't'.
- We are given this for time 0, what about time 't'?
- Inference in length-2 chain:

$$p(x_1) = \sum_{x_0} p(x_1, x_0) \quad \text{(marginalization rule)}$$

$$= \sum_{x_0} p(x_1 | x_0) p(x_0) \quad \text{(product rule)}$$

We sum over values of x_0, summing probability of leading to x_1.

Matrix notation:

Let μ_0 be a row-vector with elements $p(x_0)$.

Let P be a matrix with elements $P_{ij} = p(x_1 = j | x_0 = i)$.

Let μ_1 be a row-vector with elements $p(x_1)$.

Then $\mu_1 = \mu_0 P$.

Inference in Markov Chains

- **Inference** in length-3 chain:

$$p(x_1) = \sum_{x_0}\sum_{x_2} p(x_2, x_1, x_0) \quad \text{(marg. rule)}$$

$$= \sum_{x_0}\sum_{x_2} p(x_2 \mid x_1, x_0)\, p(x_1 \mid x_0)\, p(x_0) \quad \text{(product rule)}$$

$$= \sum_{x_0}\sum_{x_2} p(x_2 \mid x_1)\, p(x_1 \mid x_0)\, p(x_0) \quad \text{(Markov property)}$$

$$= \sum_{x_0} p(x_1 \mid x_0)\, p(x_0) \underbrace{\sum_{x_2} p(x_2 \mid x_1)}_{=1} \quad \left(\text{distributive law: } \sum_i c a_i = c \sum_i a_i\right)$$

$$= \sum_{x_0} p(x_1 \mid x_0)\, p(x_0) \quad \text{(probabilities sum to 1)}$$

Note: same as length-2 result.
- possible 'futures' do not change present.
- to get all probabilities, $u_1 = u_0 P$

Inference in Markov Chains

- **Inference** in length-3 chain:

$$p(x_2) = \sum_{x_1} \sum_{x_0} p(x_2, x_1, x_0) \quad \text{(marg. rule)}$$

$$= \sum_{x_1} \sum_{x_0} p(x_2 | x_1, x_0) p(x_1 | x_0) p(x_0) \quad \text{(prod. rule)}$$

$$= \sum_{x_1} \sum_{x_0} p(x_2 | x_1) p(x_1 | x_0) p(x_0) \quad \text{(Markov)}$$

$$= \sum_{x_1} p(x_2 | x_1) \underbrace{\sum_{x_0} p(x_1 | x_0) p(x_0)}_{p(x_1)} \quad \text{(distr. law)}$$

$$= \sum_{x_1} p(x_2 | x_1) p(x_1) \quad \text{(definition of } x_1\text{)}$$

Note: once you have $p(x_1)$ for all values of x_1, it's <u>easy</u> to compute $p(x_2)$.

In matrix notation:

$$\mu_2 = \mu_1 P \quad \text{"Chapman-Kolmogorov equation"}$$

$$= \mu_0 P P$$

$$= \mu_0 P^2$$

Similarly, $p(x_3) = \sum_{x_2} p(x_3 | x_2) p(x_2)$

$$\mu_3 = \mu_0 P^3.$$

(If inhomogeneous, $\mu_3 = \mu_0 P_1 P_2 P_3$)

Stationary Distribution of Markov Chains

- After 't' steps, the distribution of states is given by matrix power:

$$u_t = u_0 P^t$$

- After each step, we start 'forgetting' initial state.
- Stationary distribution is a steady-state:

$$u_\infty = u_\infty P$$

- A 'row' eigenvector of 'P' with an eigenvalue of 1.
  - Normalized to sum to 1.
- Often approximated using power of P (PageRank: 'power method').
- If $P_{ij} > 0$, guarantees unique stationary distribution.
  - Otherwise, could have multiple possibilities.
  - Many other conditions guarantee uniqueness.

Decoding in Markov Chains

- A subtle difference:
  - Inference let's us compute $p(x_t = s)$ for any 't' and 's'.
  - This lets us find the most likely state at each time.
  - But probability of states is dependent, may not be mostly likely sequence.
- Finding most likely sequence is called decoding:

$$\underset{x_0, x_1, \ldots, x_t}{\operatorname{argmax}} \left\{ p(x_t, x_{t-1}, \ldots, x_0) \right\}$$

- Seems harder than inference:
  - optimal sequence of length 3 may not contain optimal length 2 sequence.
  - but we can solve this problem with dynamic programming.

Decoding in Markov Chains

- Let's consider the best length-2 sequence that ends in state 'x_1':

state

$$V_1(x_1) = \max_{x_0} p(x_1, x_0) = \max_{x_0} p(x_1 | x_0) p(x_0)$$

time

"value" of best sequence up to time '1' that ends in state x_1.

- Now let's consider a length-3 sequence that ends in state 'x_2':

$$V_2(x_2) = \max_{x_1, x_0} p(x_2, x_1, x_0)$$

$$= \max_{x_1} \left\{ \max_{x_0} p(x_2, x_1, x_0) \right\} \quad \text{("max of max" trick)}$$

$\max_i ab_i = a \max_i b_i$ (for $a \geq 0$)

$$= \max_{x_1} \left\{ \max_{x_0} p(x_2 | x_1) p(x_1 | x_0) p(x_0) \right\} \quad \text{(prod. rule and Markov)}$$

Definition of $V_1(x_1)$

$$= \max_{x_1} \left\{ p(x_2 | x_1) \max_{x_0} p(x_1 | x_0) p(x_0) \right\} = \max_{x_1} \left\{ p(x_2 | x_1) V_1(x_1) \right\}$$

Decoding Markov Chains

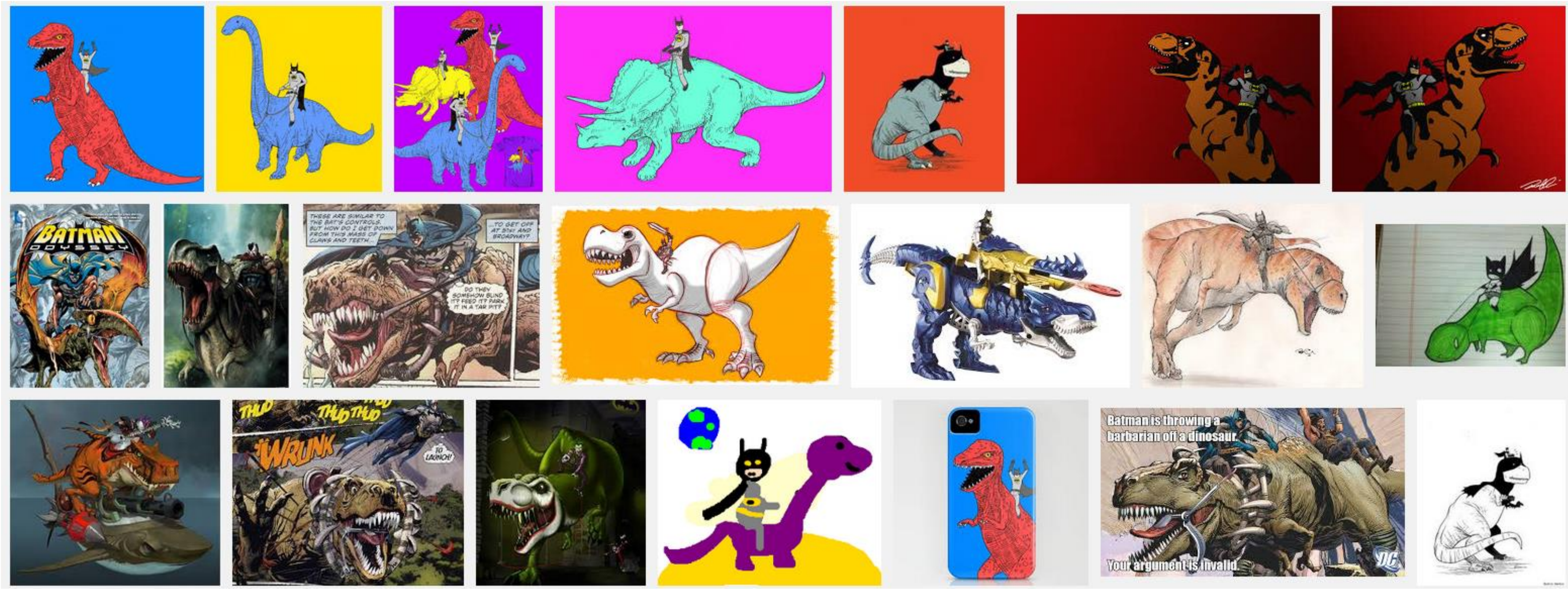
- General formula: $V_0(x_0) = p(x_0)$

$$V_t(x_t) = \max_{x_{t-1}} p(x_t | x_{t-1}) V_{t-1}(x_{t-1})$$

- We have the ingredients for dynamic programming:
  - Efficiently compute solution given recursion result:

$$\max_{x_0, x_1, \ldots, x_t} \{ p(x_t, x_{t-1}, \ldots, x_0) \} = \max_{x_t} \{ V_t(x_t) \}$$

  - Number number of possible recursions: $V_{t'}(x_{t'})$ for each time t' and state $x_{t'}$
  - If we memorize results of recursions, we can solve this efficiently.


BATMAN
ODYSSEY
THESE ARE SIMILAR TO THE BAT'S CONTROLS. BUT HOW DO I GET DOWN FROM THIS MASS OF CLAWS AND TEETH...
...TO GET OFF AT 51st AND BROADWAY?
DO THEY SOMEHOW BLIND IT? FEED IT? PARK IT IN A TAR PIT?
THUD
THUD THUD
WRUNK
TO LAUNCH!
Batman is throwing a barbarian off a dinosaur.
Your argument is invalid.

Summary

- Markov chains used for sequences/time-series/random-walks.
- Sampling is task of simulating sequence according to model.
  - Done by running a random walk.
- Learning is task of estimating parameters from sequences.
  - Done by counting.
- Inference is task of computing probabilities at particular times.
  - Done by matrix multiplication.
- Stationary distribution is steady-state after running for a long time.
  - Done by normalizing eigenvector with largest eigenvalue.
- Decoding is task of computing most likely sequence of states.
  - Done by dynamic programming.

- Next time: how Markov models are actually useful (HMMs and belief nets).