for RNN

actually not RNN. Rather why RNN needed / why other ideas don't work?

Introduction to Deep Learning

17. Sequence Models

STAT 157, Spring 2019, UC Berkeley

FRI 8/1
demo HW6 pb 1, 2

WED 8/6
demo HW6 pb 3, 4

Alex Smola and Mu Li

courses.d2l.ai/berkeley-stat-157

aws

Dependent Random Variables

Data

- So far …
  - Collect observation pairs $(x_i, y_i) \sim p(x, y)$ for training
  - Estimate $y \mid x \sim p(y \mid x)$ for unseen $x' \sim p(x)$
- Examples
  - Images & objects
  - Regression problem
  - House & house prices
- **The order of the data did not matter**

Recall - Interaction with Environment

- **Batch** (download a book)
  Observe training data (x_1,y_1) ... (x_l,y_l) then deploy
- **Online** (follow the class)
  Observe x, predict f(x), observe y (stock market, homework)
- **Active learning** (ask questions in class)
  Query y for x, improve model, pick new x
- **Bandits** (do well at homework)
  Pick arm, get reward, pick new arm (also with context)
- **Reinforcement Learning** (play chess, drive a car)
  Take action, environment responds, take new action

Recall - Stateful Systems

Recall - Training ≠ Testing

- **Generalization performance**
  (the empirical distribution lies)
- **Covariate shift**
  (the covariate distribution lies)
- **Logistic regression**
  (tools to fix shift)
- **Covariate shift correction**
- **Label shift**
  (the label distribution lies)
- **Nonstationary Environments**

$$p_{\text{emp}}(x, y) \neq p(x, y)$$

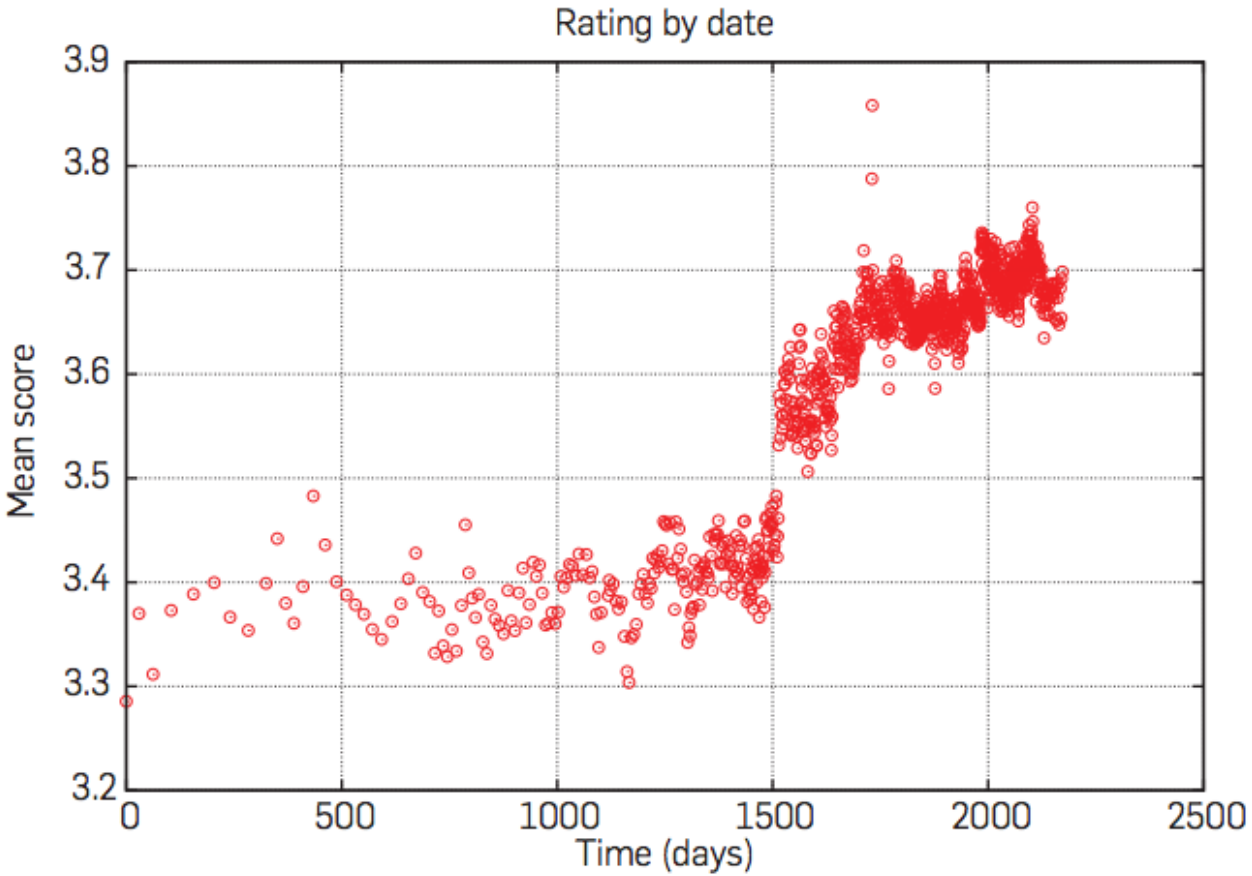
$$p(x) \neq q(x)$$

$$\log(1 + \exp(-yf(x)))$$

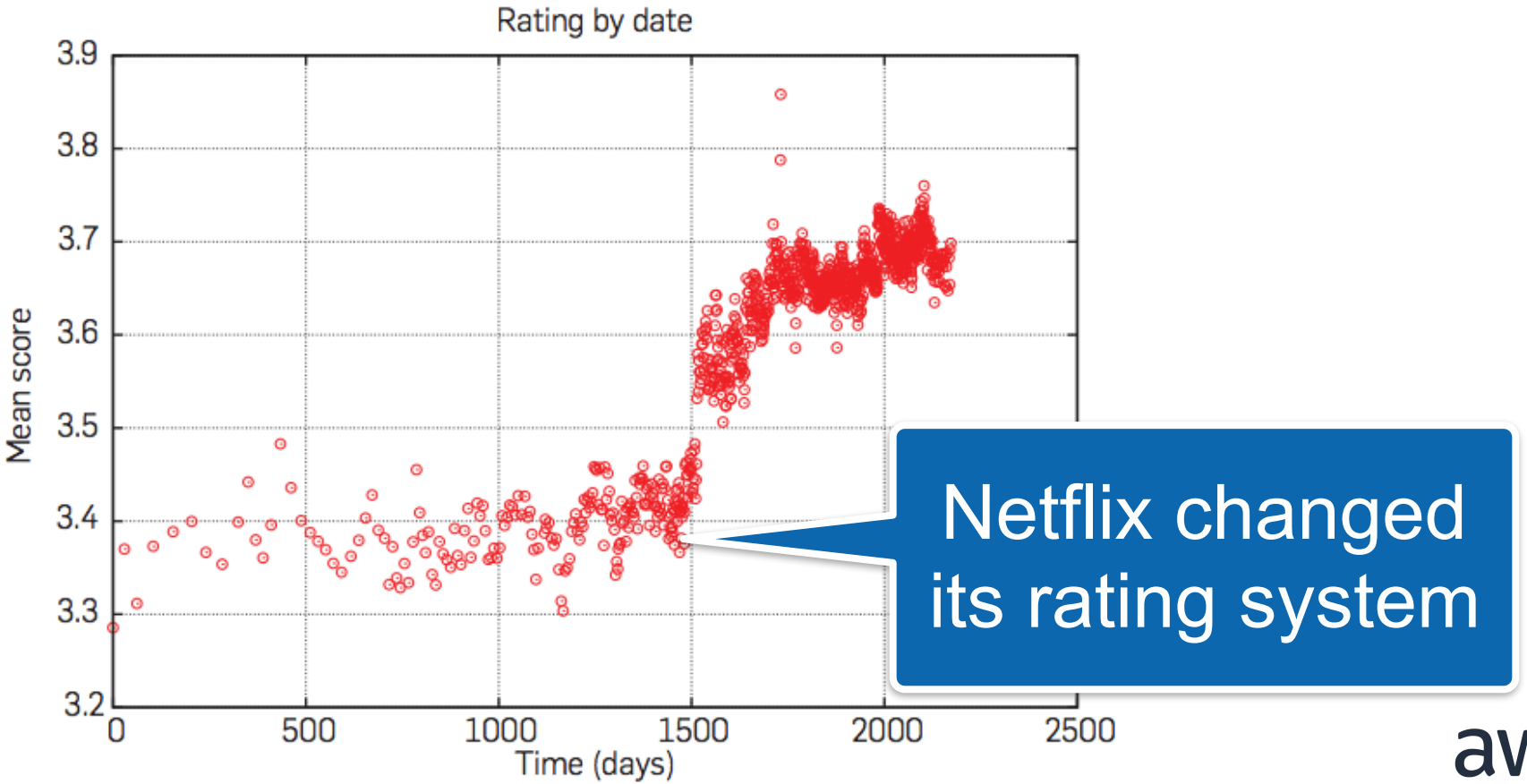
$$\frac{1}{2}\left(p(x)\delta(1, y) + q(x)\delta(-1, y)\right)$$

$$p(y) \neq q(y)$$

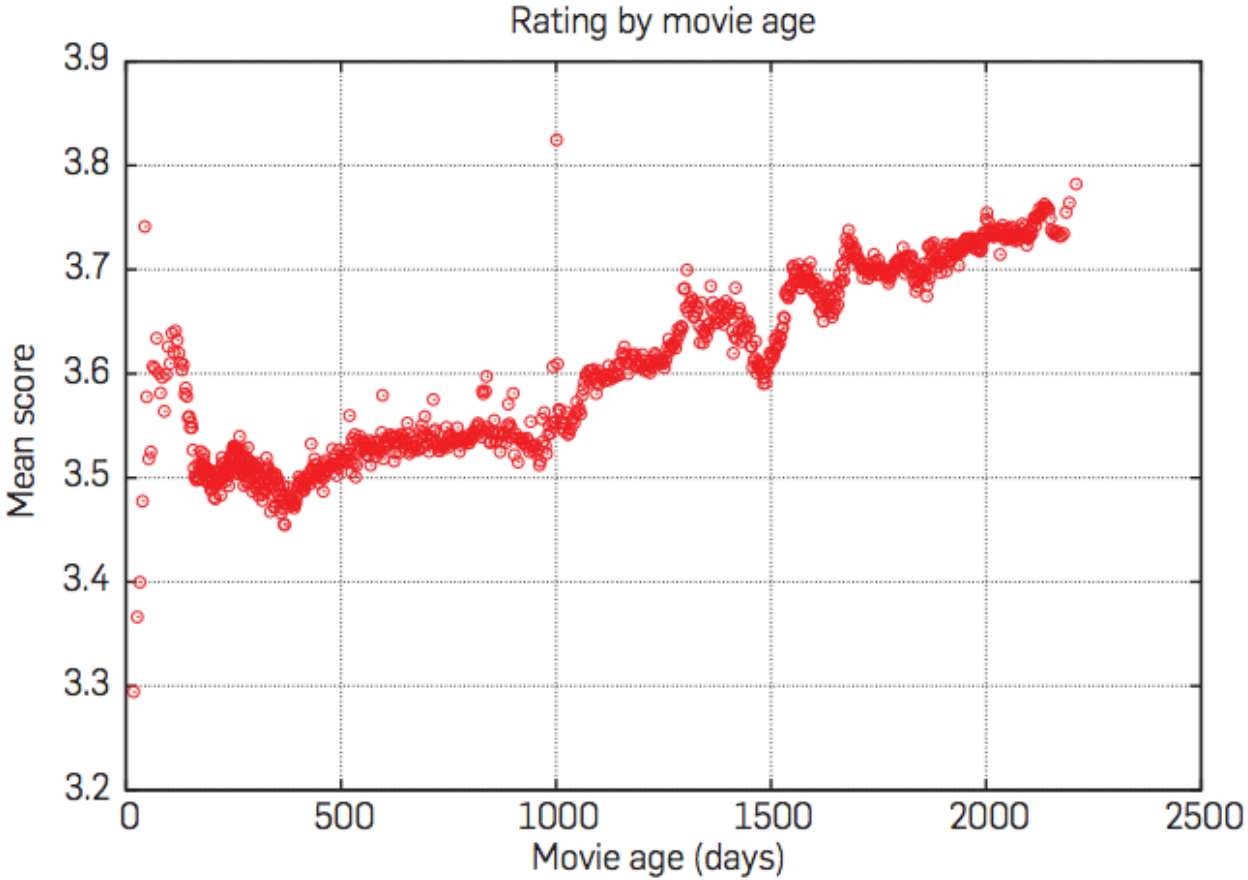
Time matters (Koren, 2009)



Time matters (Koren, 2009)



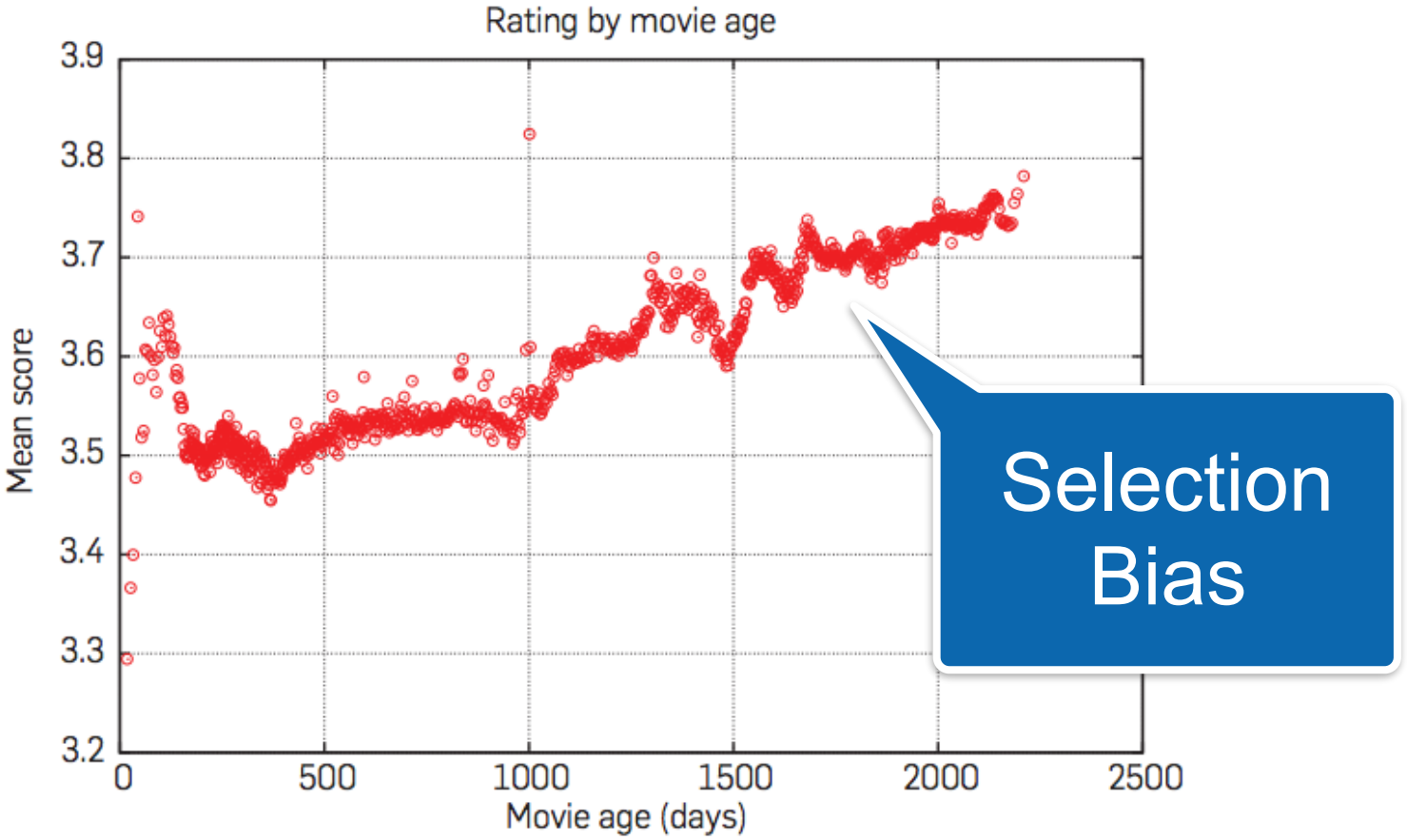
Time matters (Koren, 2009)



courses.d2l.ai/berkeley-stat-157

aws

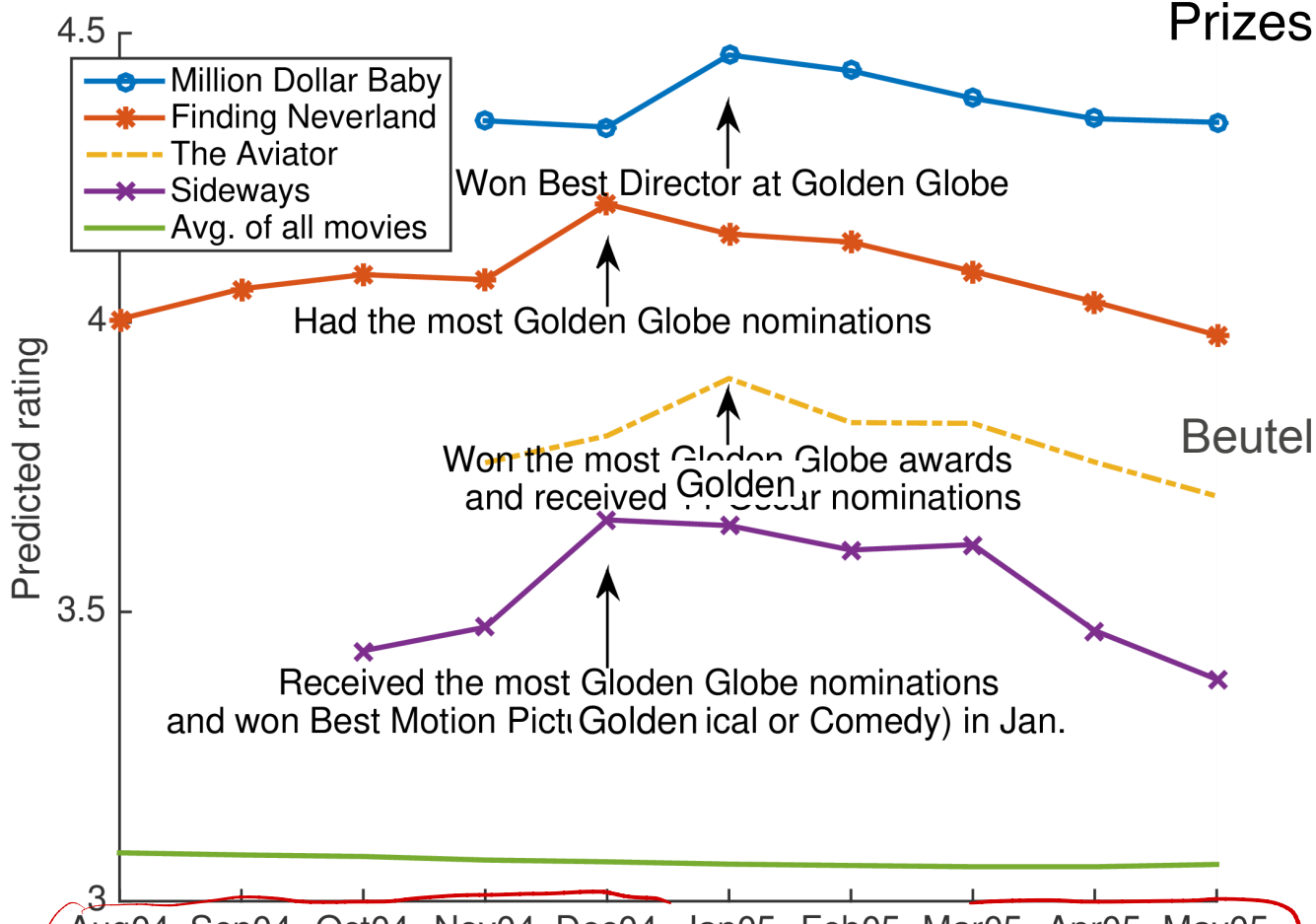
Time matters (Koren, 2009)



courses.d2l.ai/berkeley-stat-157

aws

Prizes

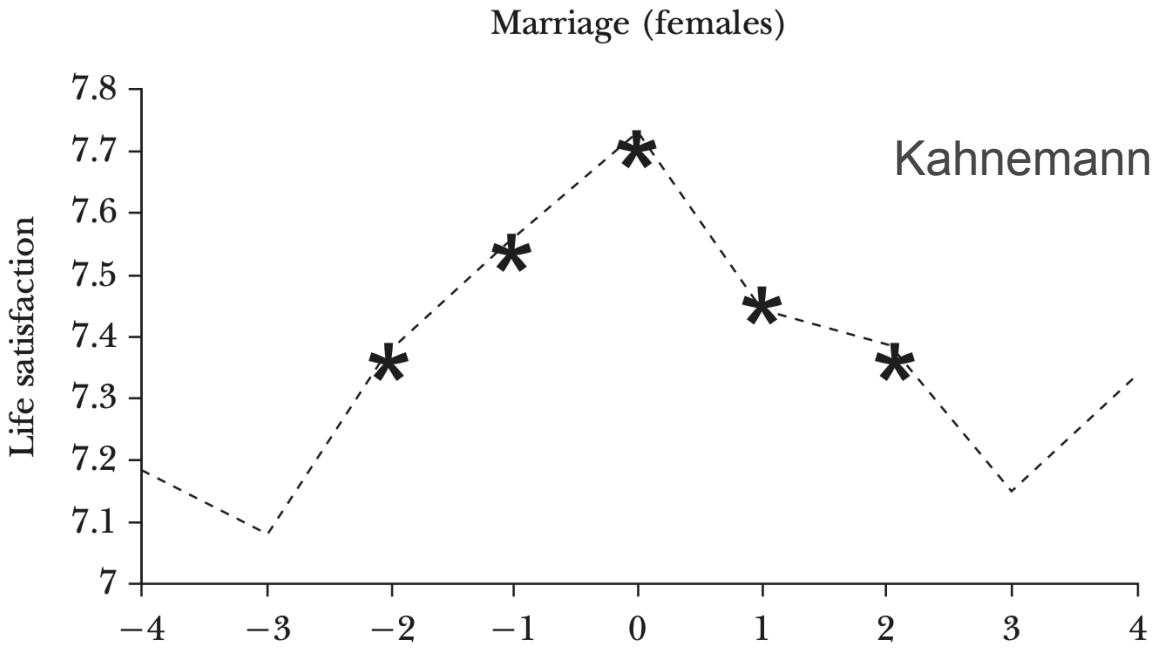


Beutel et al., 2015

five sequence

Average Life Satisfaction for a Sample of German Women

(by year of marriage $t = 0$)

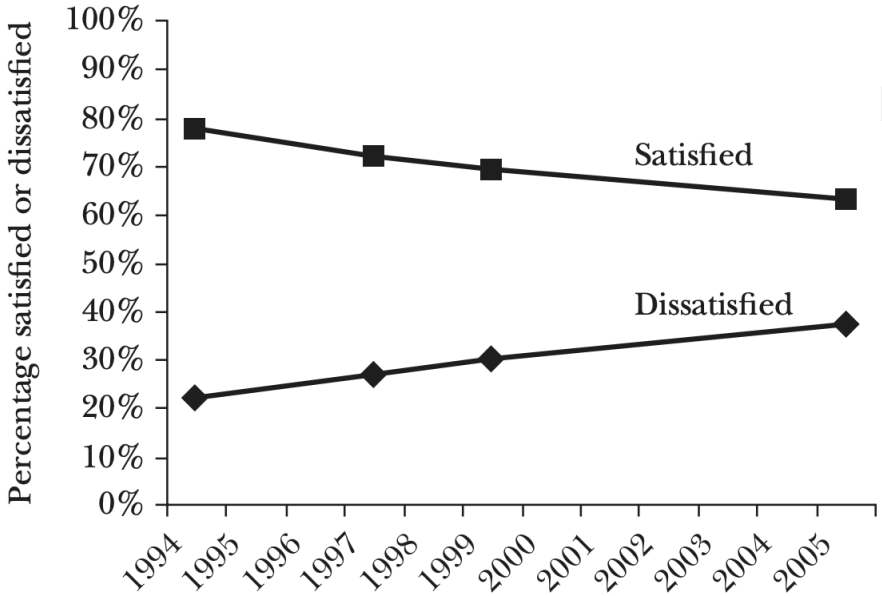


Kahnemann & Krueger, 2006

Source: Clark, Diener, Georgellis and Lucas (2003), using data from the German Socioeconomic Panel.
Note: An asterisk indicates that life satisfaction is significantly different from the baseline level.

Life Satisfaction in China as Average Real Income Rises by 250 Percent

Overall, how satisfied or dissatisfied are you with the way things are going in your life today? Would you say you are very satisfied, somewhat satisfied, somewhat dissatisfied, or very dissatisfied?

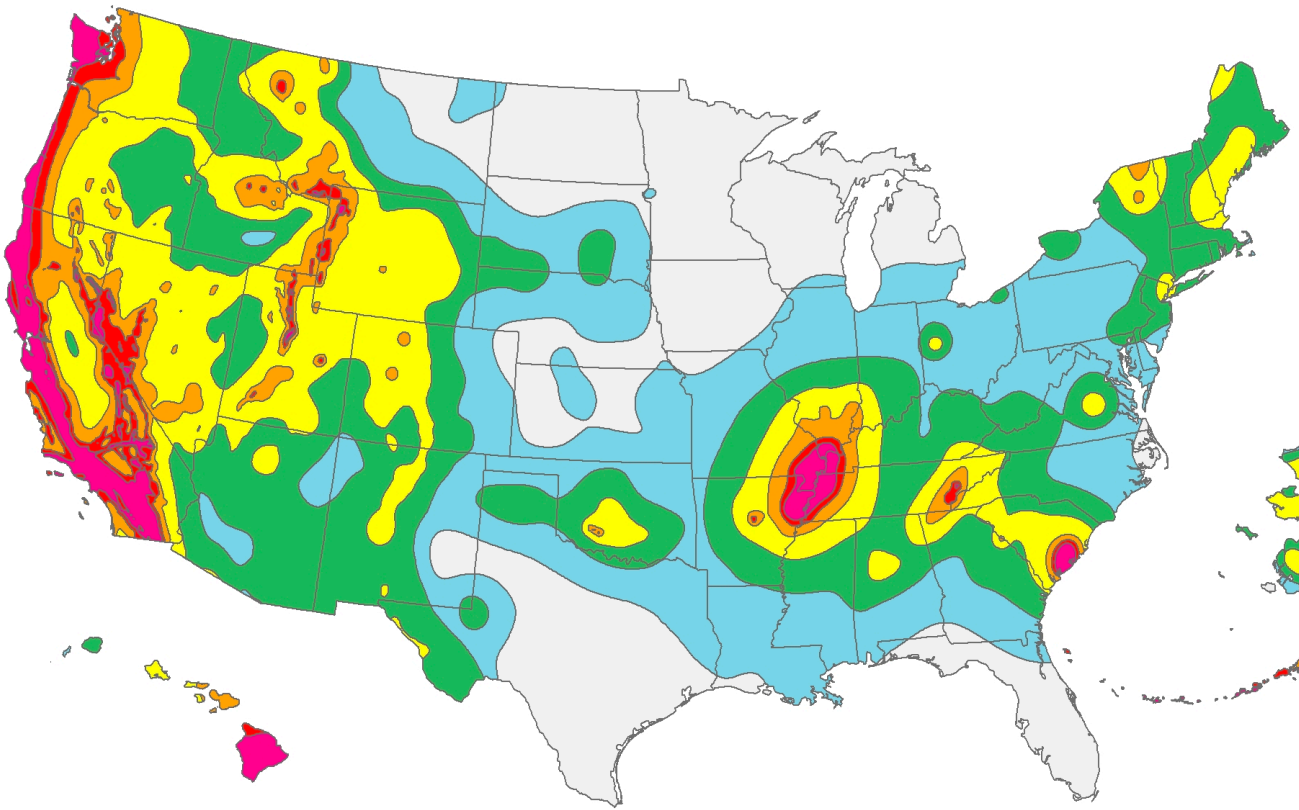


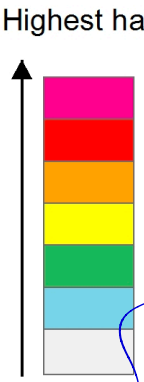
Kahnemann & Krueger, 2006

Source: Derived from Richard Burkholder, "Chinese Far Wealthier Than a Decade Ago—but Are They Happier?" The Gallup Organization, ⟨http://sww.gallup.com/poll/content/login.aspx?ci=14548⟩.

USGS
science for a changing world

Not just time. Space, too





weather prediction

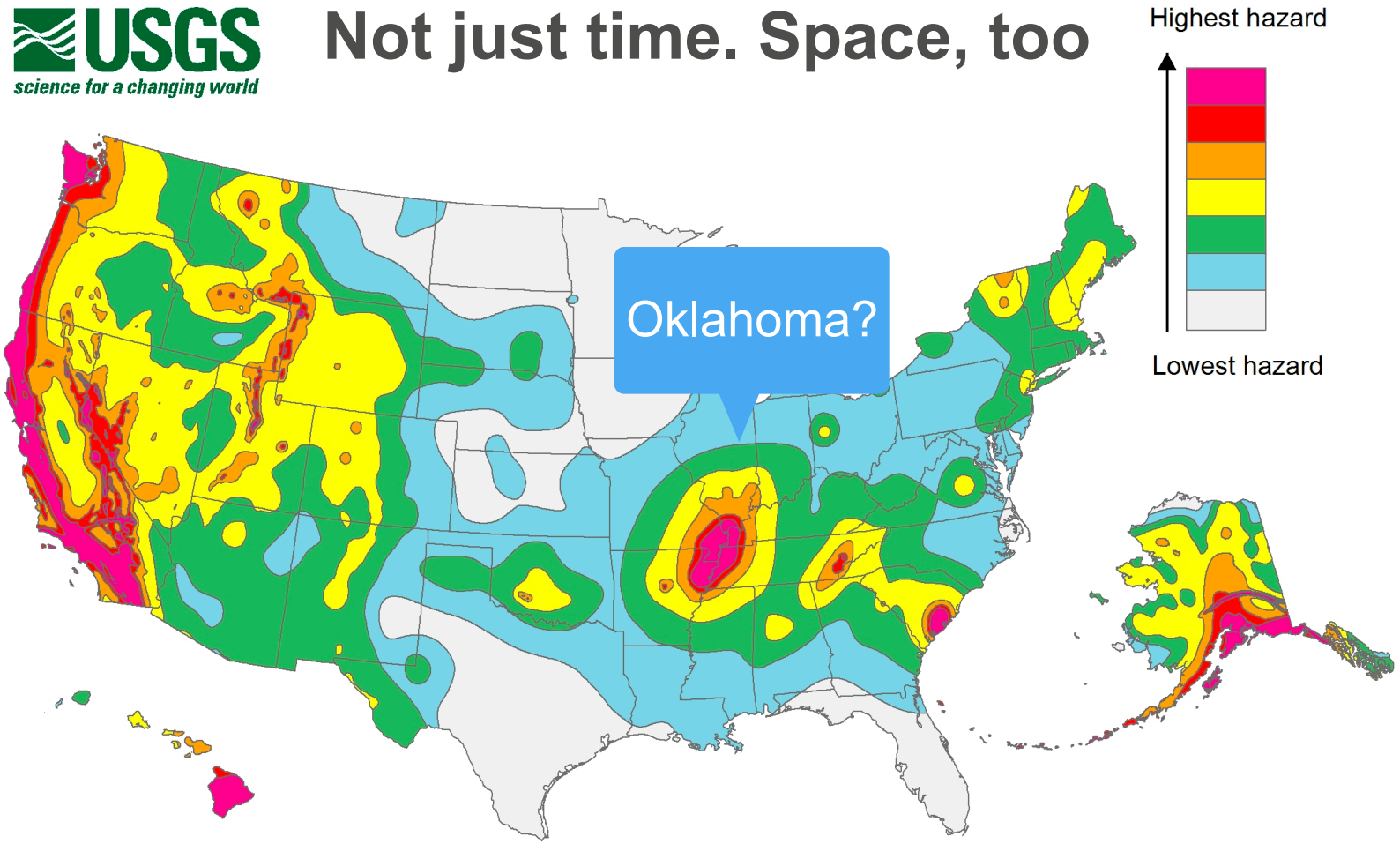
- good in short term
- terrible in long term.

long sequence of events

aws

USGS
science for a changing world

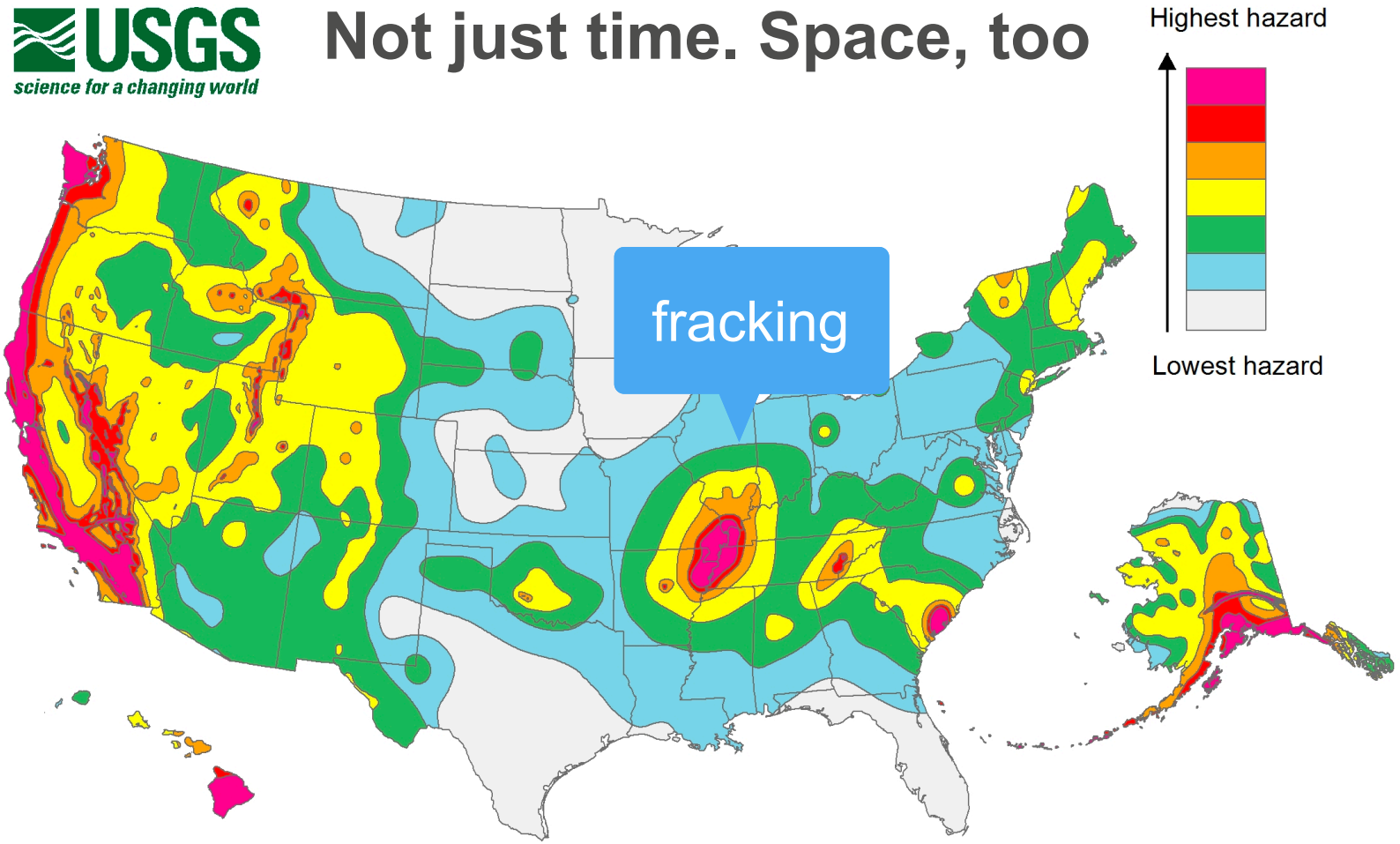
Not just time. Space, too



aws

USGS
science for a changing world

Not just time. Space, too



aws

FTSE 100

courses.d2l.ai/berkeley-stat-157

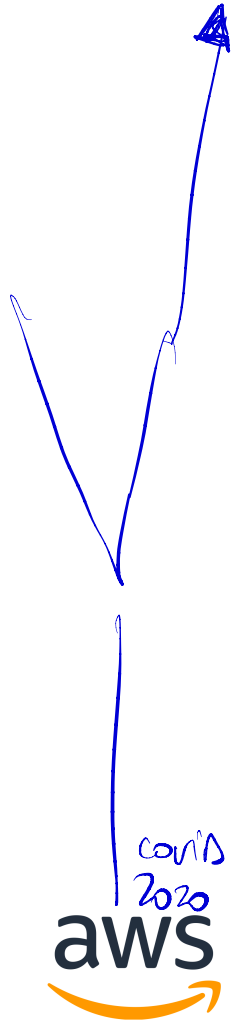
aws

FTSE 100

courses.d2l.ai/berkeley-stat-157

aws

FTSE 100

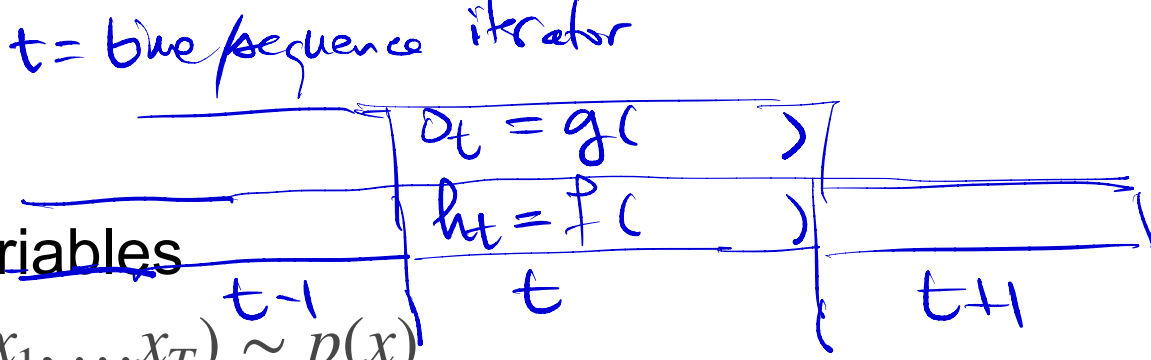


courses.d2l.ai/berkeley-stat-157

TL;DR - Data usually isn’t IID

Sequence Models

Sequence Model



- Dependent random variables

$$(x_1, \ldots x_T) \sim p(x)$$

- Conditional probability expansion

$$p(x) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_1, x_2) \cdot \ldots p(x_T \mid x_1, \ldots x_{T-1})$$

x_T given prev $x_1, x_2 \ldots x_{T-1}$

- Can always find this expansion

Bayes in order

x_2 given x_1

- Could also find reverse direction …

x_3 given x_1, x_2

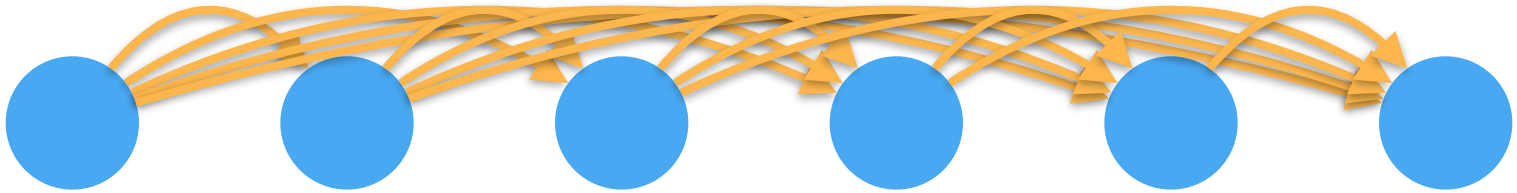
$$p(x) = p(x_T) \cdot p(x_{T-1} \mid x_T) \cdot p(x_{T-2} \mid x_{T-1}, x_T) \cdot \ldots p(x_1 \mid x_2, \ldots x_T)$$

Full back (all the way to t=1)

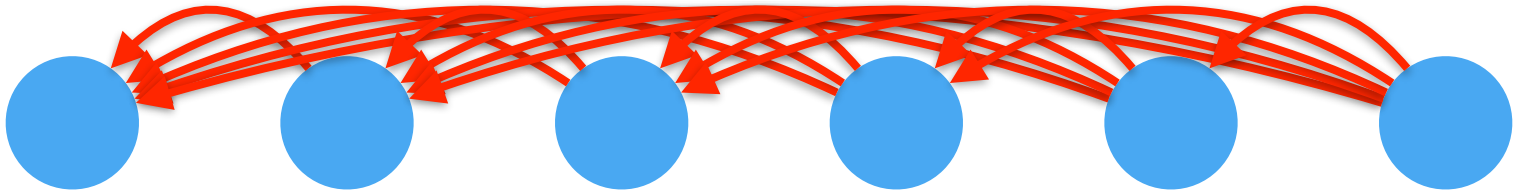
So why bother?

Sequence Model

$$p(x) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_1, x_2) \cdot \ldots p(x_T \mid x_1, \ldots x_{T-1})$$



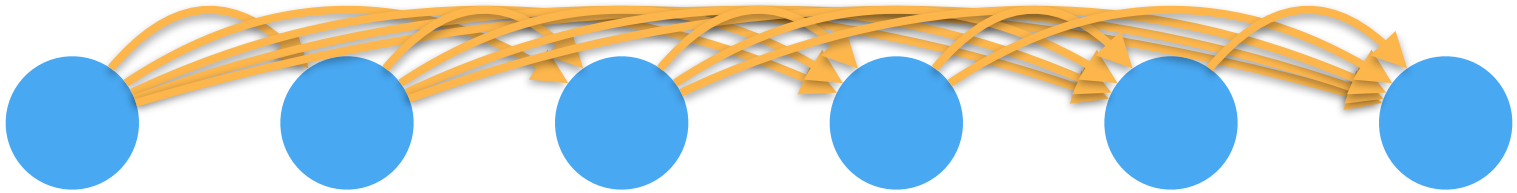
$$p(x) = p(x_T) \cdot p(x_{T-1} \mid x_T) \cdot p(x_{T-2} \mid x_{T-1}, x_T) \cdot \ldots p(x_1 \mid x_2, \ldots x_T)$$



- Causality (physics) prevents the reverse direction
- 'wrong' direction often much more complex to model

Sequence Model

$$p(x) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_1, x_2) \cdot \ldots p(x_T \mid x_1, \ldots x_{T-1})$$



- Autoregressive model

$$p(x_t \mid x_1, \ldots x_{t-1}) = p(x_t \mid f(x_1, \ldots x_{t-1}))$$

model $p(x_t)$ as a model "f" = regression of previous steps

current

past

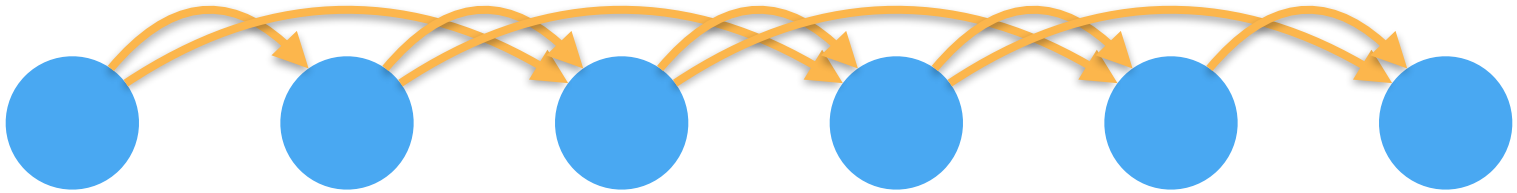
Some function of previously seen data

Plan A - Markov Assumption

$x_t \leftarrow x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}$

limited memory; ex last 4 steps (not further)

$$p(x) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_1, x_2) \cdot \ldots p(x_T \mid x_{T-\tau}, \ldots x_{T-1})$$



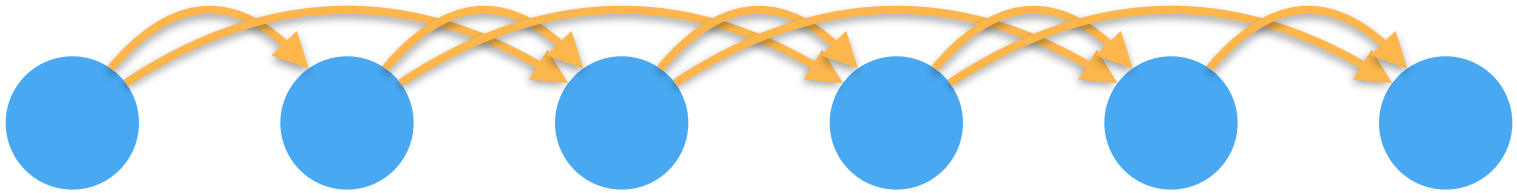
- Assume that only a few steps in the past matter
- Autoregressive model

$$p(x_t \mid x_1, \ldots x_{t-1}) = p(x_t \mid f(x_{t-\tau}, \ldots x_{t-1}))$$

Some function of previously seen data

Plan A - Markov Assumption

$$p(x) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_1, x_2) \cdot \ldots p(x_T \mid x_{T-\tau}, \ldots x_{T-1})$$



- In practice solve regression problem

$\tau = 4$

$$\hat{x}_t = f(x_{t-\tau}, \ldots x_{t-1}) = f(x_{t-4}, x_{t-3}, x_{t-2}, x_{t-1})$$

e.g. train an MLP on previously seen data

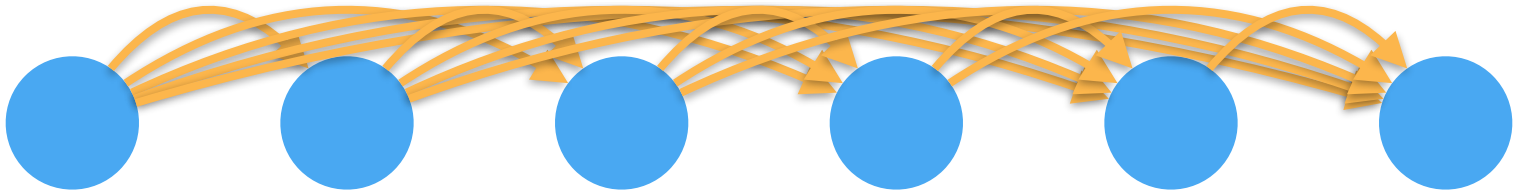
Plan B - Latent Variable Model

Alternative

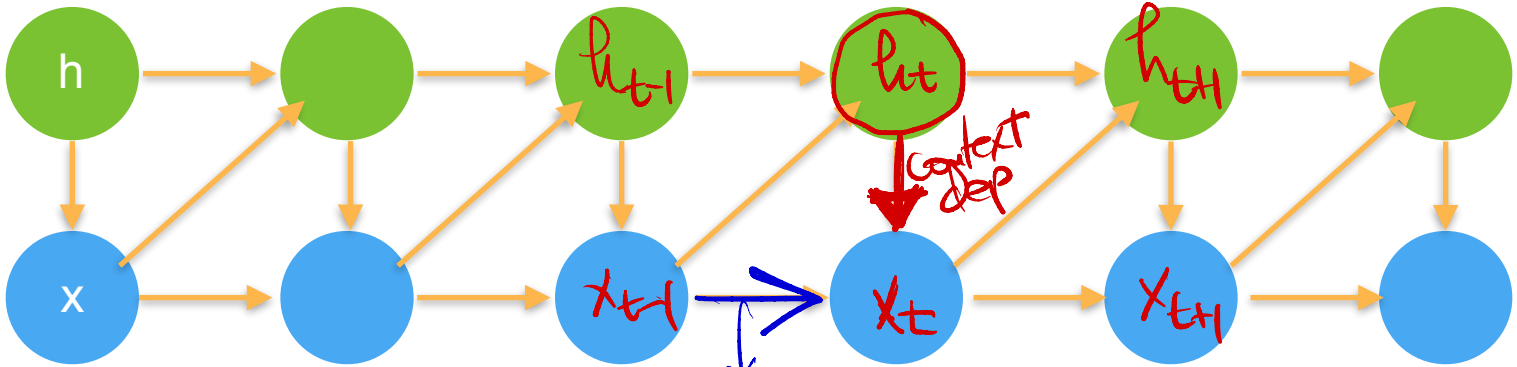
h_t = hidden state at step t

= context at time t

$$p(x) = p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_1, x_2) \cdot \ldots p(x_T \mid x_1, \ldots x_{T-1})$$



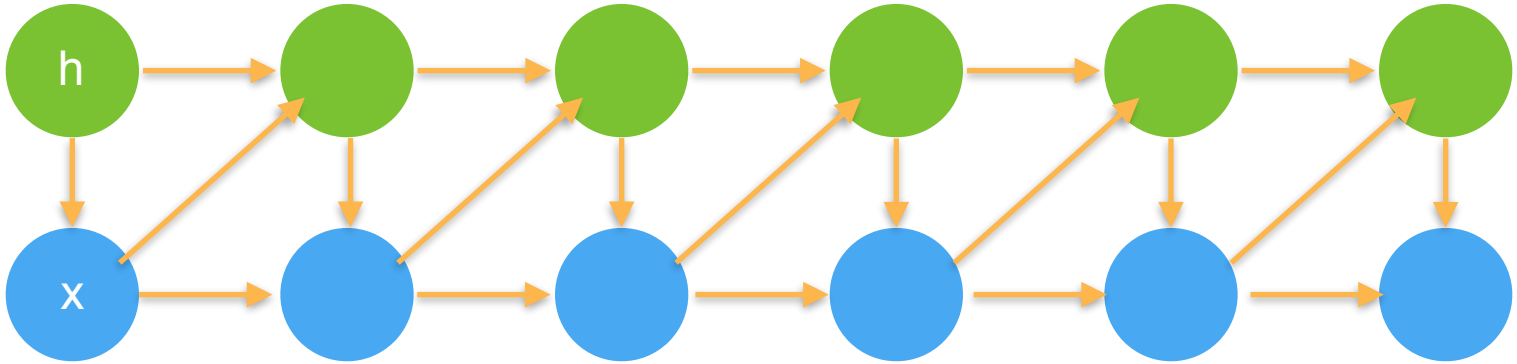
$p(h_t \mid h_{t-1}, x_{t-1})$ and $p(x_t \mid h_t, x_{t-1})$



Plan B - Latent Variable Model

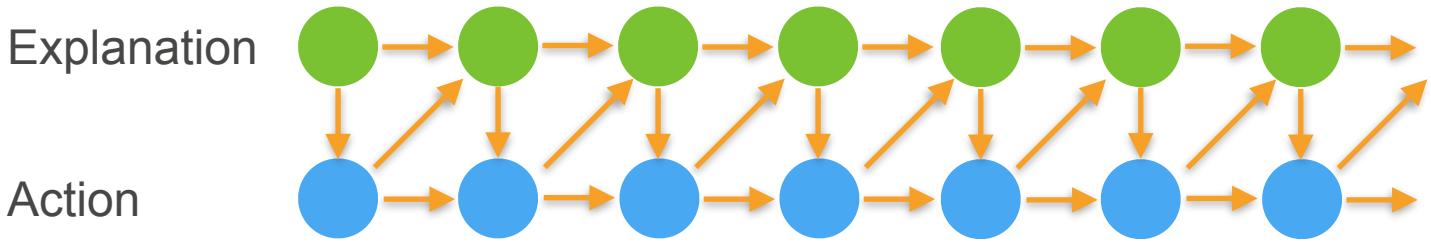
- Latent state summarizes all the relevant information about the past. So we get $h_t = f(x_1, \ldots x_{t-1}) = f(h_{t-1}, x_{t-1})$

$p(h_t \mid h_{t-1}, x_{t-1})$ and $p(x_t \mid h_t, x_{t-1})$



Latent Variable Models (classical treatment)

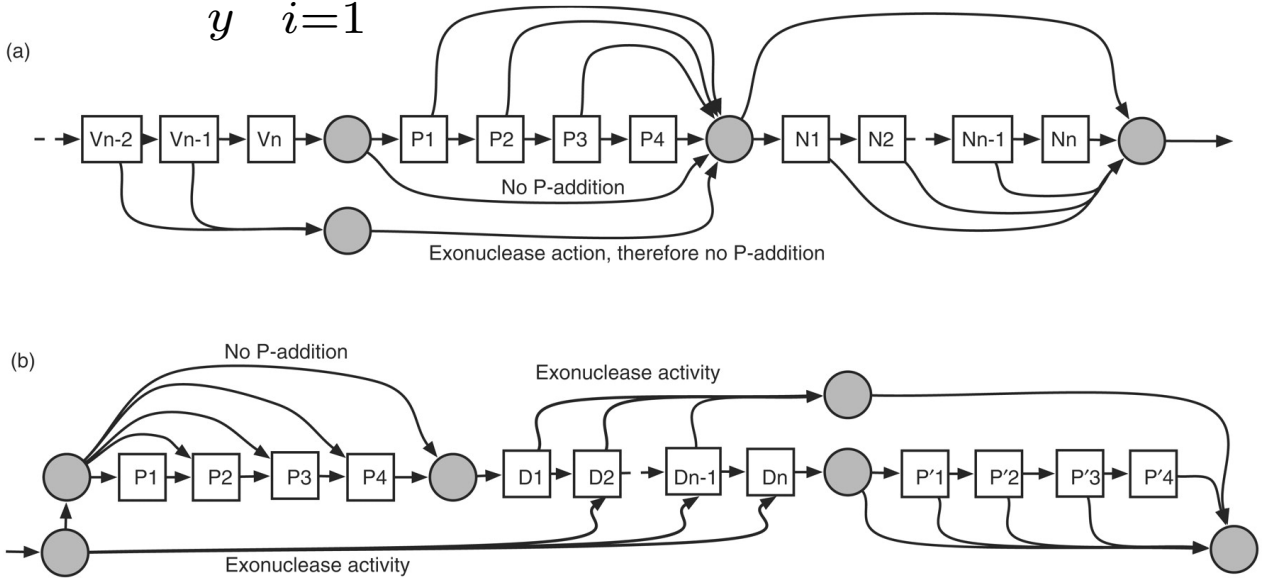
- **Temporal sequence of observations**
  Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**
  - Clusters (navigational, informational queries in search)
  - Topics (interest distributions for users over time)
  - Kalman Filter (trajectory and location modeling)



Temporal Clustering aka Hidden Markov Models

- Clusters with sequential dependence

$$p(x) = \sum_{y} \prod_{i=1}^{n} p(y_i|y_{i-1})p(x_i|y_i)$$

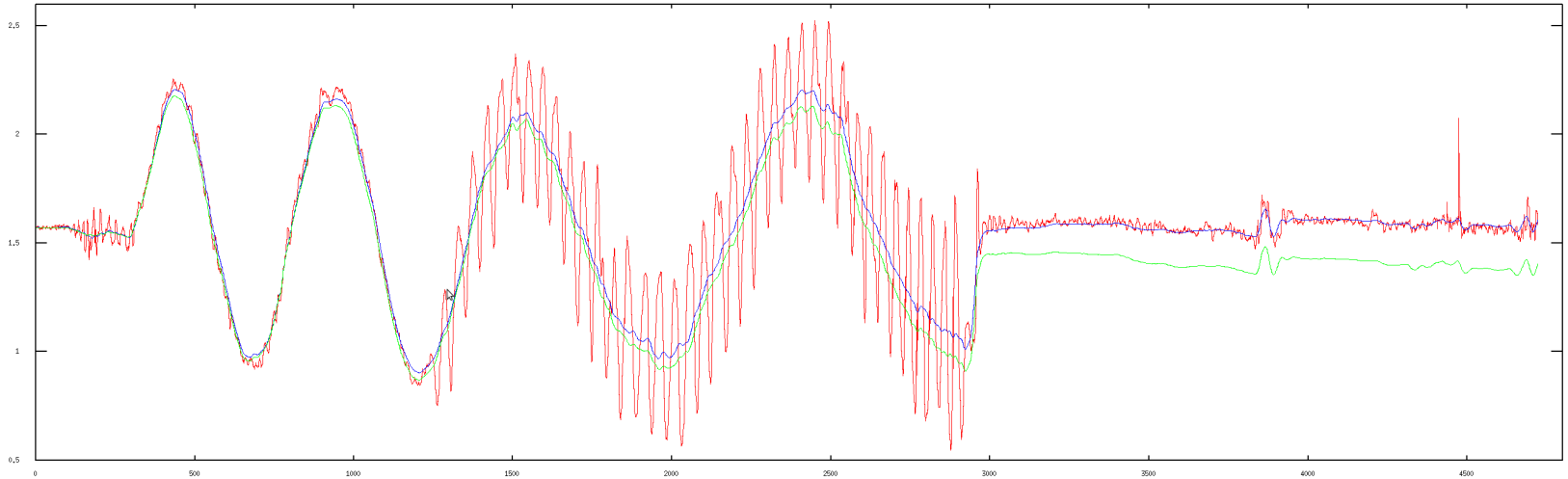


Temporal PCA aka Kalman Filter

- Latent factor variable
- Simple sequential factorial structure

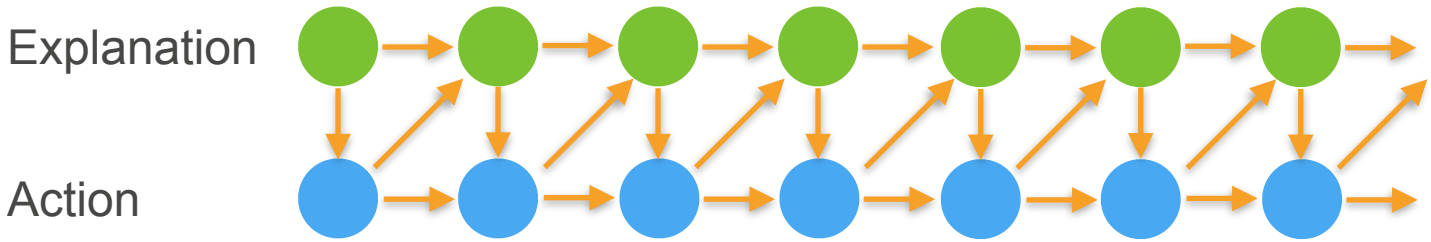
$$x_t \sim \mathcal{N}(Ay_t + \mu, K)$$

$$y_t \sim \mathcal{N}(By_{t-1} + \nu, L)$$



Sequence Models

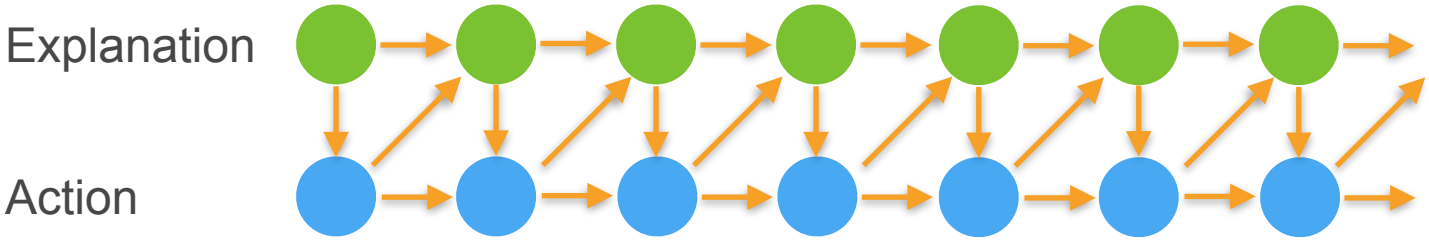
- **Temporal sequence of observations**
  Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**
  - Clusters (navigational, informational queries in search)
  - Topics (interest distributions for users over time)
  - Kalman Filter (trajectory and location modeling)



Sequence Models

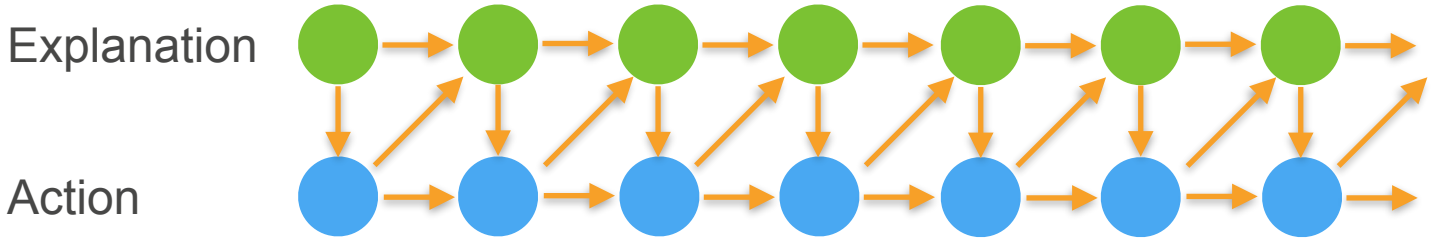
- **Temporal sequence of observations**
  Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**

Are the parametric models really true?



Sequence Models

- **Temporal sequence of observations**
  Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **State space**
  Variable depth / variable representations / variable types
  (we know more about some users, queries than others)
- **Temporal resolution**
  Data doesn't arrive at quantized intervals



Plan A - Markov Assumption

Markov Assumption

- Next observation only depends on the past few terms

$$\hat{x}_t = f(x_{t-\tau}, \ldots x_{t-1})$$

- Train regression model
- Use it to predict the next step and iterate

Demo time

Language Models

text/language → sequence

for a particular problem

- predict the continuation of sentence
- translation | text-seq-english → text-seq-spanish

HW6

Modeling Language 101

- Tokens not real values (domain is countably finite)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_1, \ldots, w_{t-1})$$

$$\begin{aligned} & p(\text{Statistics}, \text{is}, \text{fun}, .) \\ = \; & p(\text{Statistics})\, p(\text{is} \mid \text{Statistics})\, p(\text{fun} \mid \text{Statistics}, \text{is})\, p(. \mid \text{Statistics}, \text{is}, \text{fun}) \end{aligned}$$

- Estimating it

$$\hat{p}(\text{is} \mid \text{Statistics}) = \frac{n(\text{Statistics is})}{n(\text{Statistics})}$$

N-grams (longer sequences of tokens)

- Need smoothing (long n-grams are infrequent)

$$\hat{p}(w) = \frac{n(w) + \epsilon_1/m}{n + \epsilon_1}$$

$$\hat{p}(w' \mid w) = \frac{n(w, w') + \epsilon_2 \hat{p}(w')}{n(w) + \epsilon_2}$$

$$\hat{p}(w'' \mid w', w) = \frac{n(w, w', w'') + \epsilon_3 \hat{p}(w', w'')}{n(w, w') + \epsilon_3}$$

Hack

Talk to your friendly Bayesian if you want to do it right!

Let’s look at actual language statistics

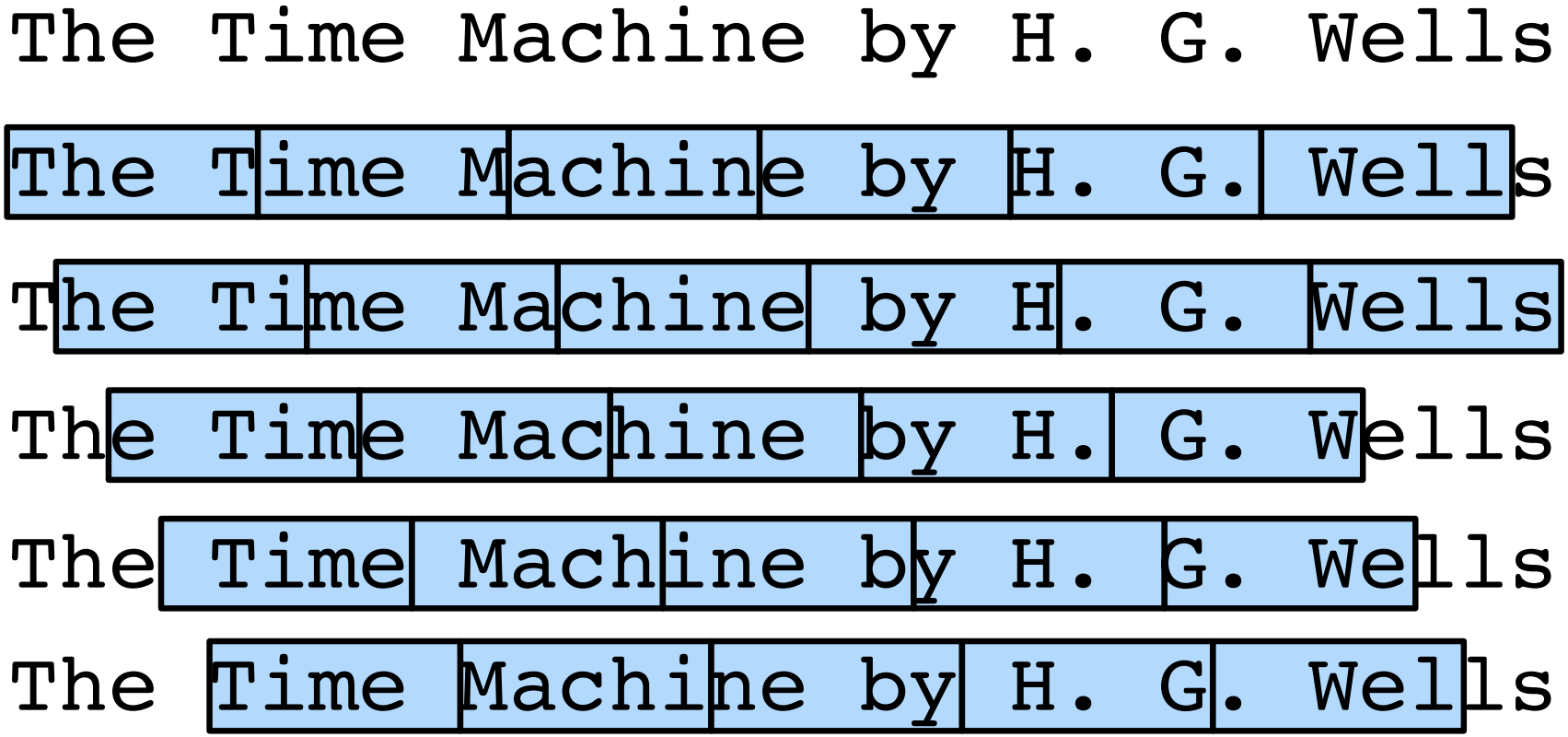
Text Preprocessing

Tokenization

`The Time Machine by H. G. Wells`

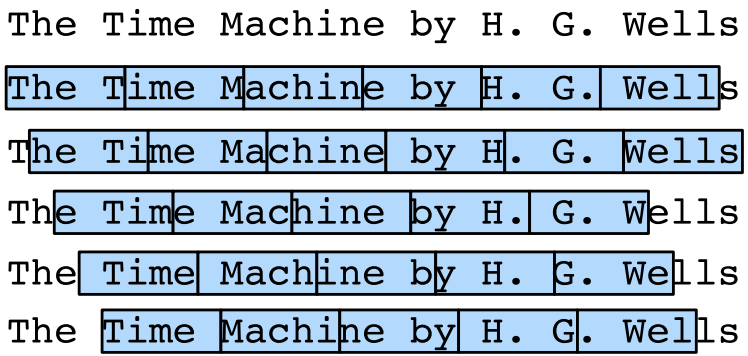
- Basic Idea - map text into sequence of IDs
- **Character Encoding** (each character has one ID)
  - Small vocabulary
  - Doesn’t work so well (DNN needs to learn spelling)
- **Word Encoding** (each word has one ID)
  - Accurate spelling
  - Doesn’t work so well (huge vocabulary = costly multinomial)
- **Byte Pair Encoding** (Goldilocks zone)
  - Frequent subsequences (like syllables)

Minibatch Generation



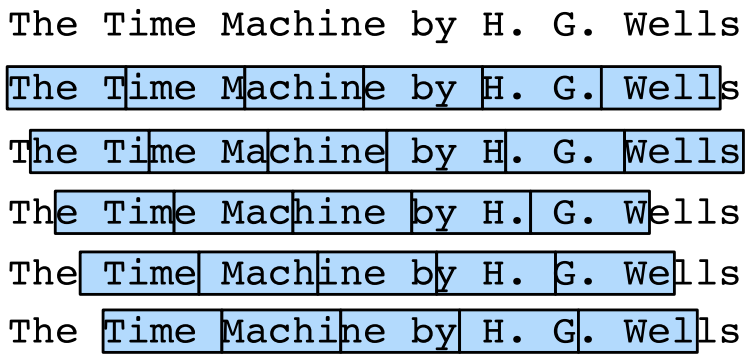
Minibatch Generation

- **Random partitioning**
  - Pick random offset
  - Distribute sequences at random over mini batches
  - Independent-ish samples
  - Need to reset hidden state



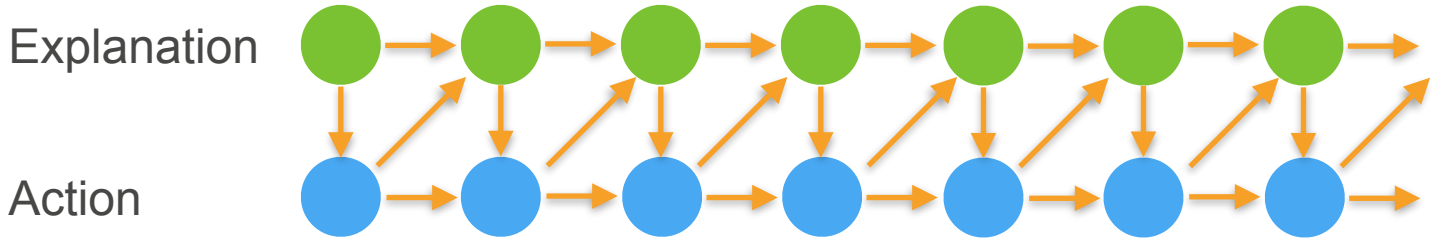
Minibatch Generation

- **Sequential partitioning**
  - Pick random offset
  - Distribute sequences in sequence over mini batches
  - Dependent samples
  - Keep hidden state across mini batches (much better)

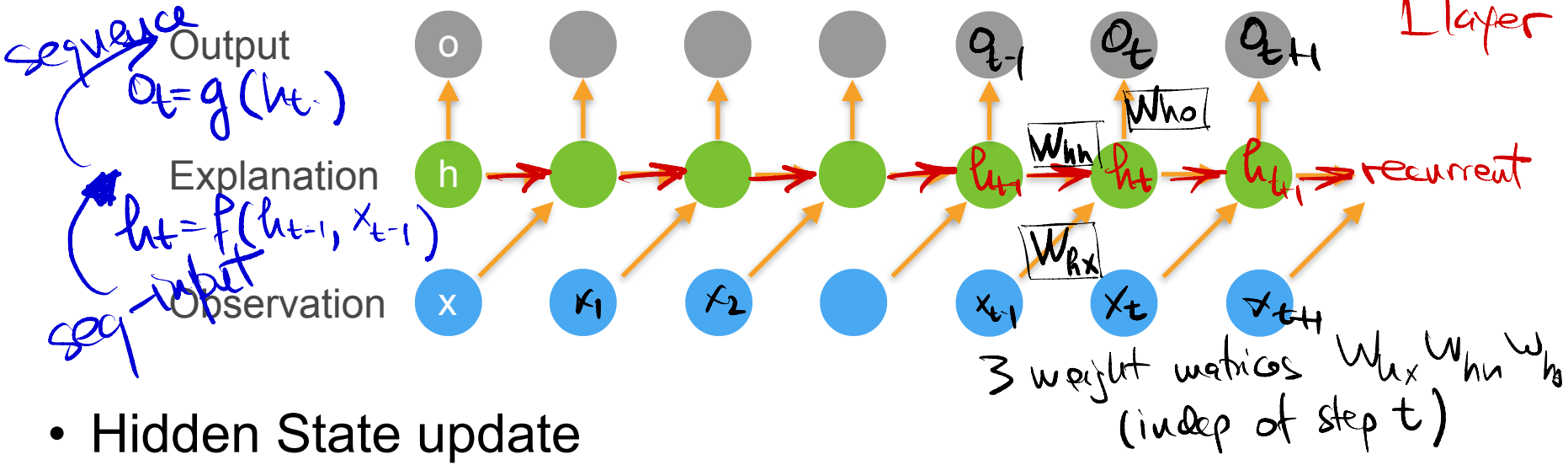


Plan B - Recurrent Neural Networks

Recurrent Neural Networks (with hidden state)



Recurrent Neural Networks (with hidden state)



- Hidden State update

$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

- Observation update

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

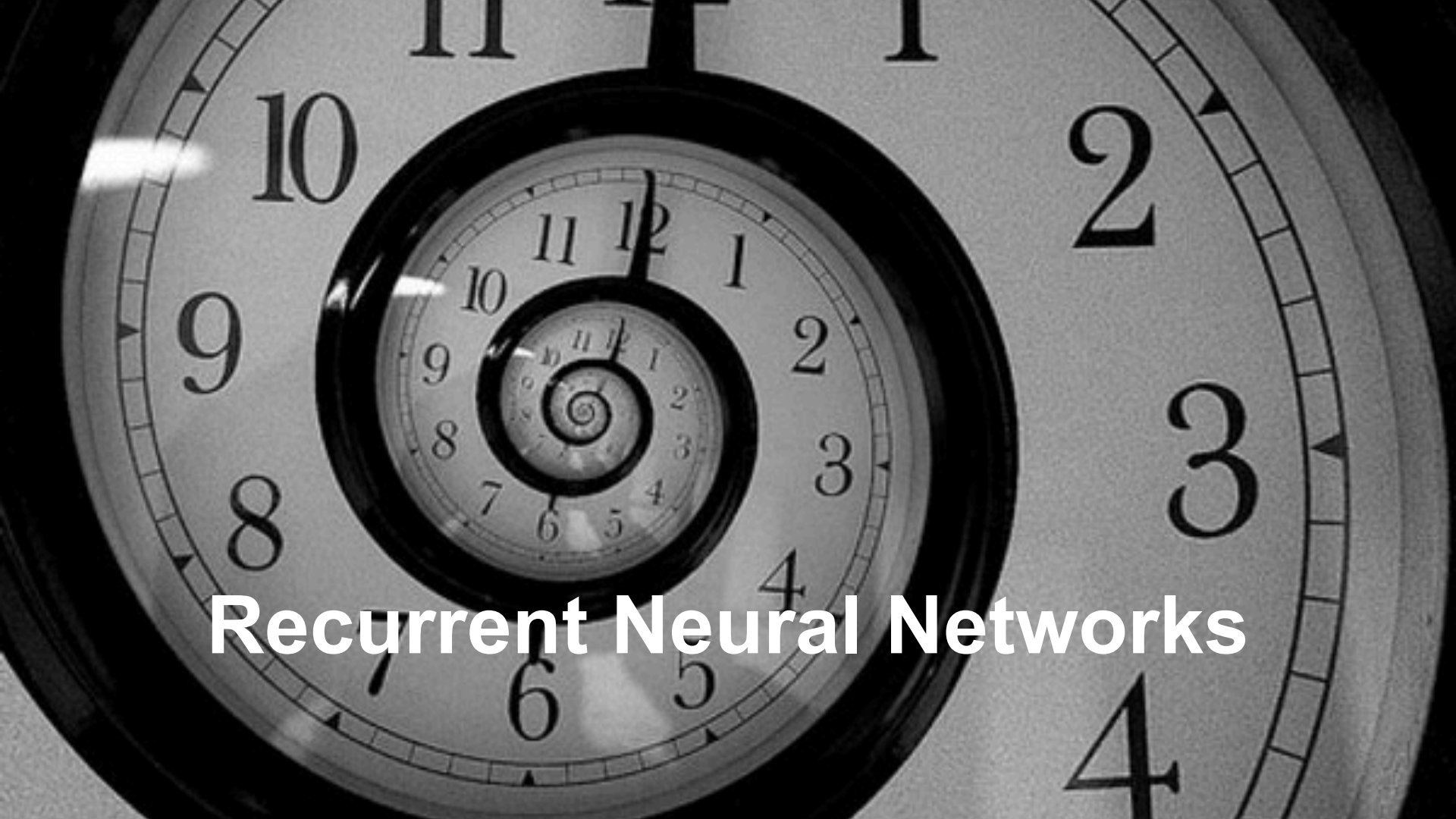
Introduction to Deep Learning

19. Recurrent Neural Networks

STAT 157, Spring 2019, UC Berkeley

Alex Smola and Mu Li

courses.d2l.ai/berkeley-stat-157

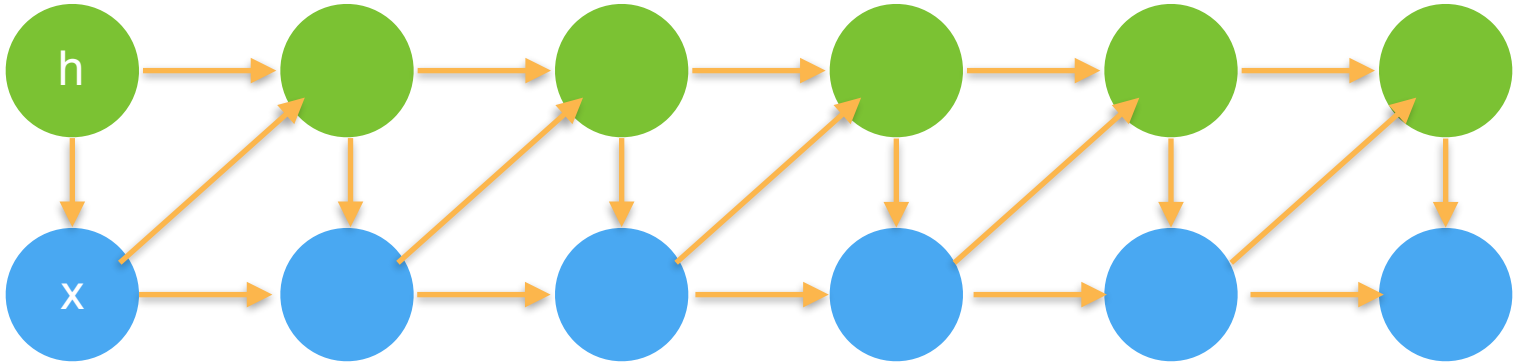


Recurrent Neural Networks

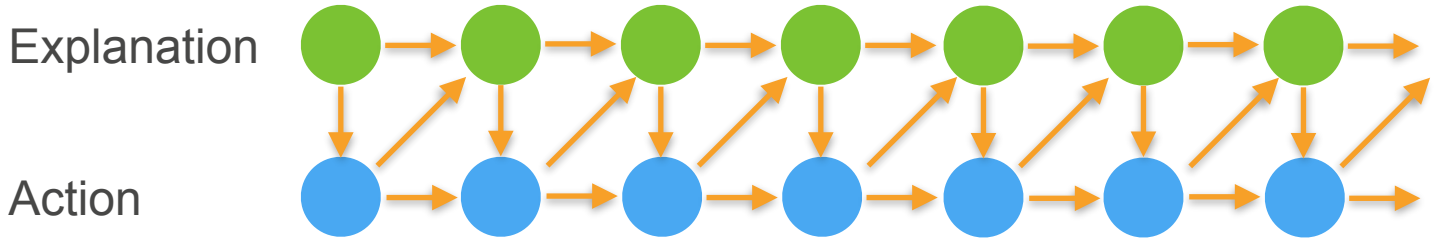
Latent Variable Autoregressive Models

Latent state summarizes all the relevant information about the past. So we get $h_t = f(x_1, \ldots x_{t-1}) = f(h_{t-1}, x_{t-1})$

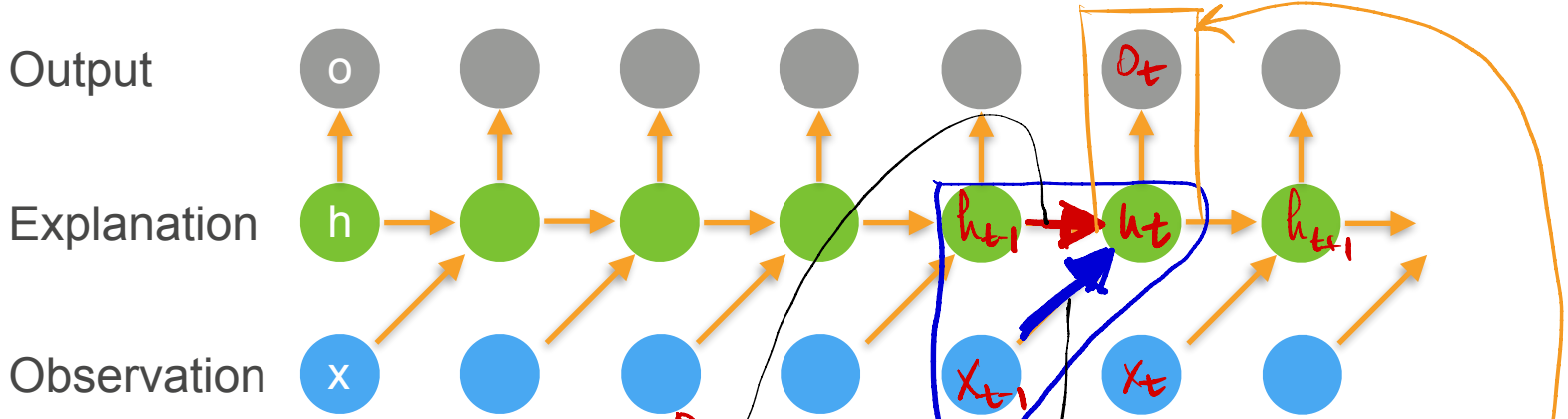
$p(h_t \mid h_{t-1}, x_{t-1})$ and $p(x_t \mid h_t, x_{t-1})$



Recurrent Neural Networks (with hidden state)



Recurrent Neural Networks (with hidden state)



- Hidden State update

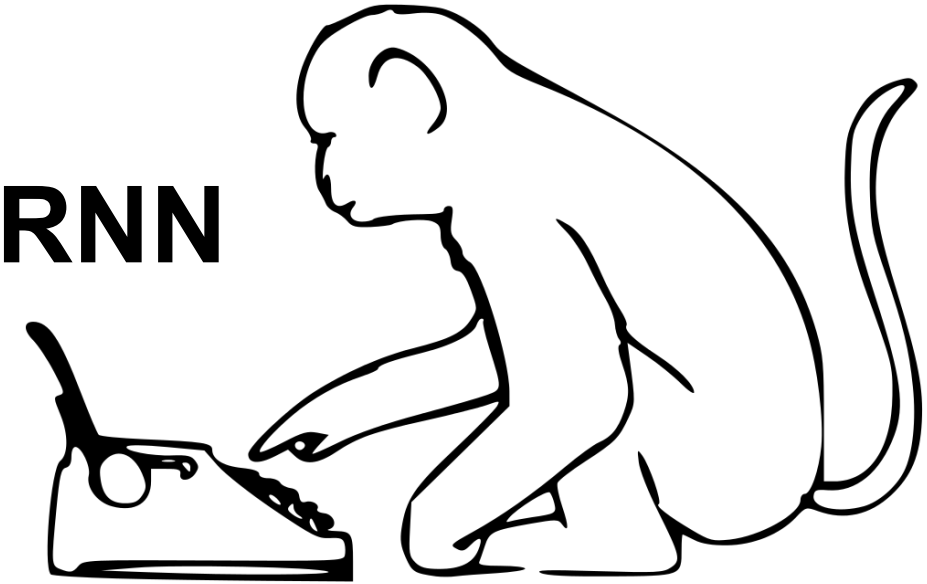
$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

- Observation update

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

Code …

Implementing an RNN Language Model



Input Encoding

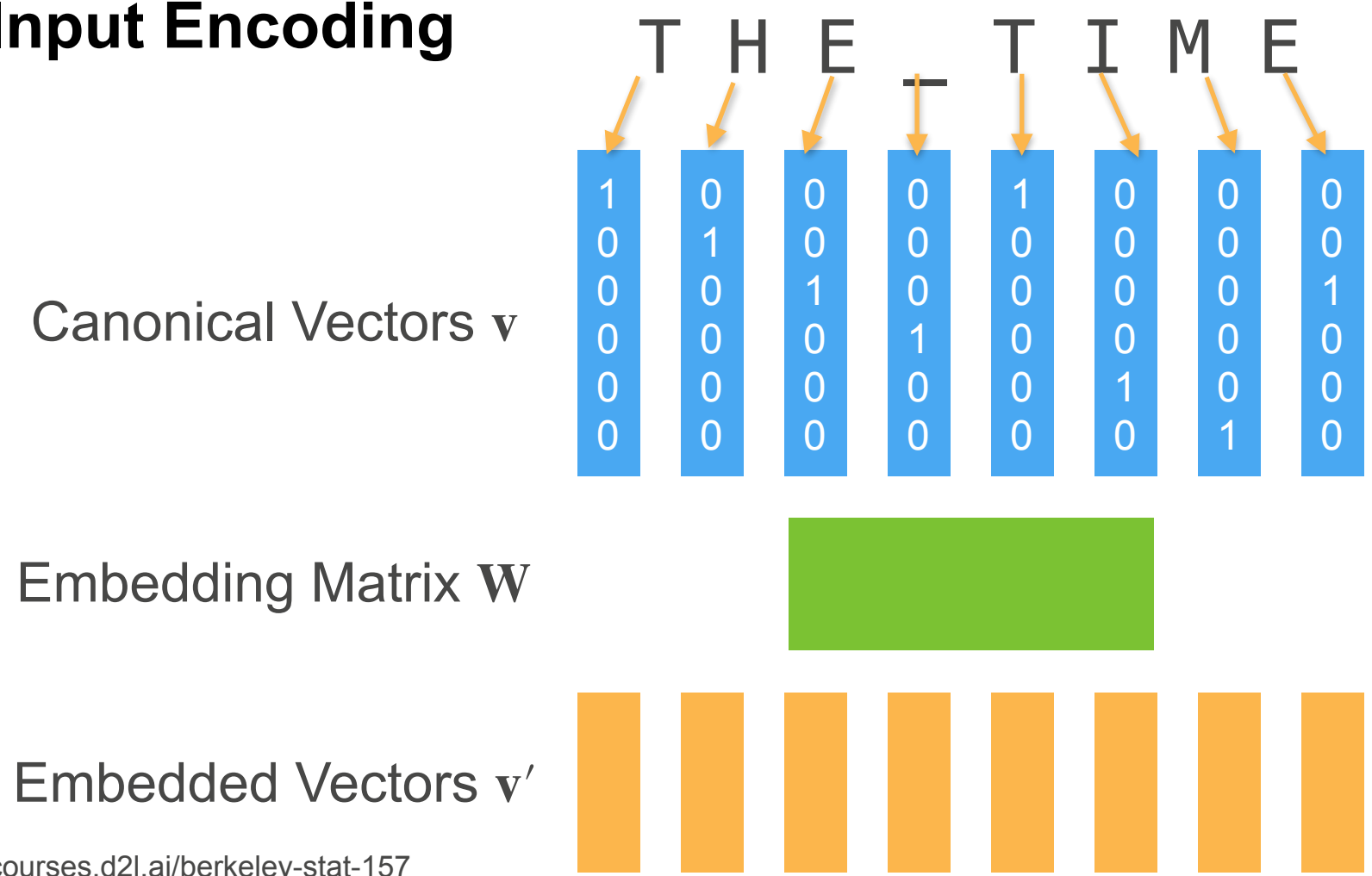
- Need to map input tokens to vectors
  - Pick granularity (words, characters, subwords)
  - Map to indicator vectors

```
nd.one_hot(nd.array([0, 2]), vocab_size)
```

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
<NDArray 2x43 @cpu(0)>
```

- Multiply by embedding matrix W

Input Encoding



RNN with hidden state mechanics

- Input
  vector sequence $\mathbf{x}_1, \ldots, \mathbf{x}_T$
- Hidden States
  vector sequence $\mathbf{h}_1, \ldots, \mathbf{h}_T$ where $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$
- Output
  vector sequence $\mathbf{o}_1, \ldots, \mathbf{o}_T$ where $\mathbf{o}_t = g(\mathbf{h}_t)$

  Read sequence to generate hidden states, then start generating outputs. Often outputs (symbols) are used as input for next hidden state (and thus output).

Output Decoding

Output Vectors $\mathbf{o}$

Decoder

Decoding Matrix $\mathbf{W}'$

Encoder

$$p(y \mid \mathbf{o}) \propto \exp\left(\mathbf{v}_y^\top \mathbf{o}\right) = \exp(\mathbf{o}[y])$$

One-hot decoding

Gradients

- Long chain of dependencies for backprop
  - Need to keep a lot of intermediate values in memory
  - Butterfly effect style dependencies
  - Gradients can vanish or diverge (more on this later)
- Clipping to prevent divergence

$$\mathbf{g} \leftarrow \min\left(1, \frac{\theta}{\|\mathbf{g}\|}\right)\mathbf{g}$$

rescales to gradient of size at most θ

Perplexity = exp^Entropy

- Typically measure accuracy with log-likelihood
  - This makes outputs of different length incomparable (e.g. bad model on short output has higher likelihood than excellent model on very long output)
  - Normalize log-likelihood to sequence length

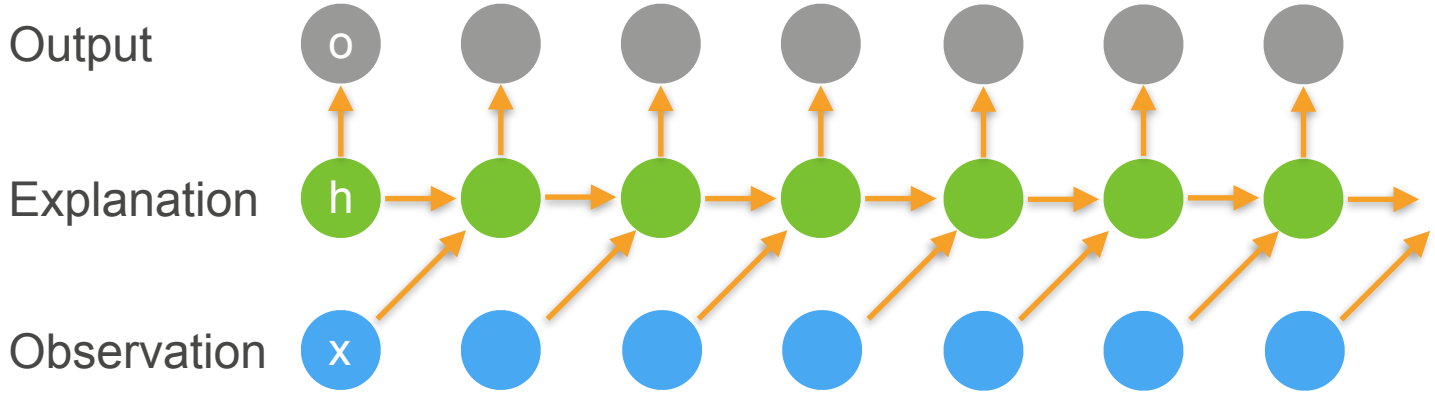
comparable with var. lengths

$$\log(\Pi \ldots) = -\sum_{t=1}^{T} \log p(y_t | \text{model}) \quad \text{vs.} \quad \pi := -\frac{1}{T} \sum_{t=1}^{T} \log p(y_t | \text{model})$$

- Perplexity is exponentiated version $\exp(\pi)$ (effectively number of possible choices on average)


Truncated Backprop Through Time

Recurrent Neural Networks (with hidden state)



- Hidden State update

$$h_t = f(h_{t-1}, x_{t-1}, w)$$

- Observation update

$$o_t = g(h_t, w)$$

Objective function

- RNN generates output which needs to be compared to target labels

$$L(x, y, w) = \sum_{t=1}^{T} l(y_t, o_t)$$

- Gradient

$$\partial_w L = \sum_{t=1}^{T} \partial_w l(y_t, o_t)$$

$$= \sum_{t=1}^{T} \partial_{o_t} l(y_t, o_t) \left[\partial_w g(h_t, w) + \partial_{h_t} g(h_t, w) \partial_w h_t \right]$$

chain rule

Latent State Gradient $\partial_w h_t$

grad (h_t) w.r.t w_{hh}

- Objective Function

$$\partial_w L = \sum_{t=1}^{T} \partial_w l(y_t, o_t) = \sum_{t=1}^{T} \partial_{o_t} l(y_t, o_t) \left[\partial_w g(h_t, w) + \partial_{h_t} g(h_t, w) \partial_w h_t \right]$$

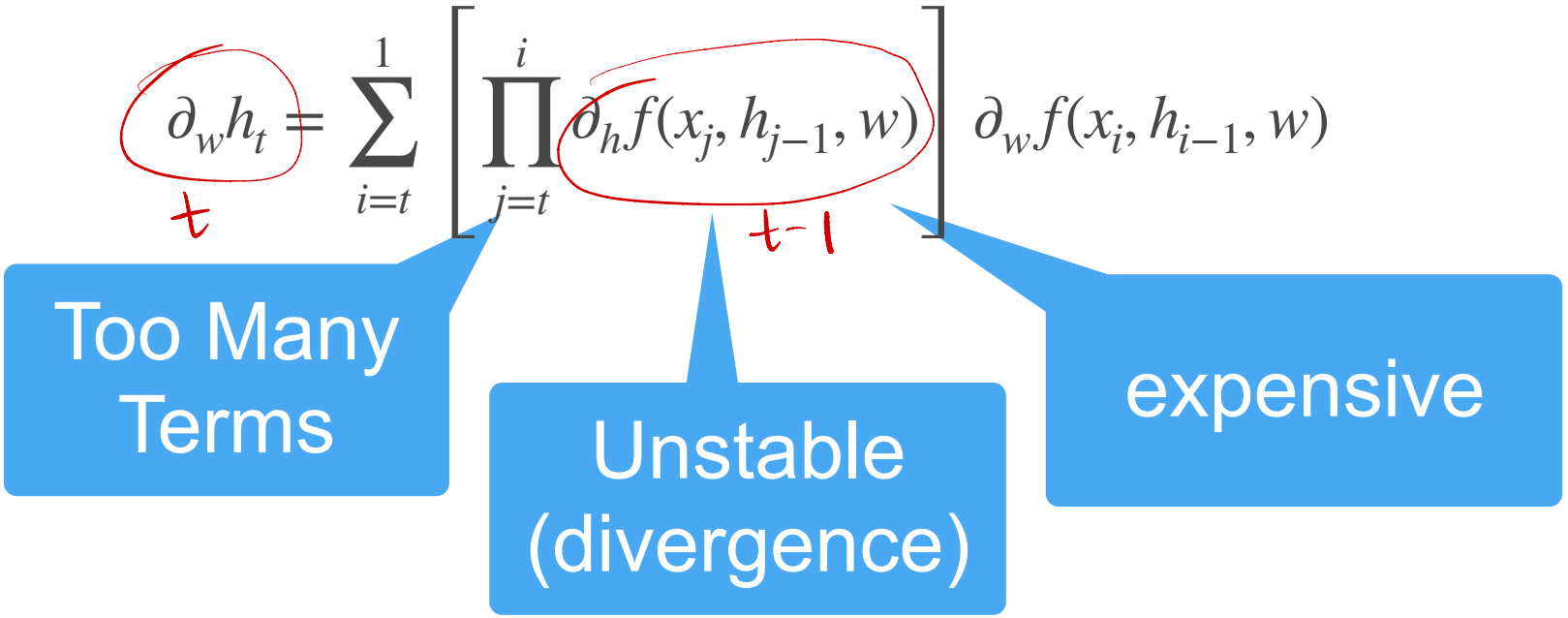
- Gradient Recursion

$$\partial_w h_t = \partial_w f(x_t, h_{t-1}, w) + \partial_h f(x_t, h_{t-1}, w) \partial_w h_{t-1}$$

$$= \sum_{i=t}^{1} \left[\prod_{j=t}^{i} \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$

Latent State Gradient $\partial_w h_t$

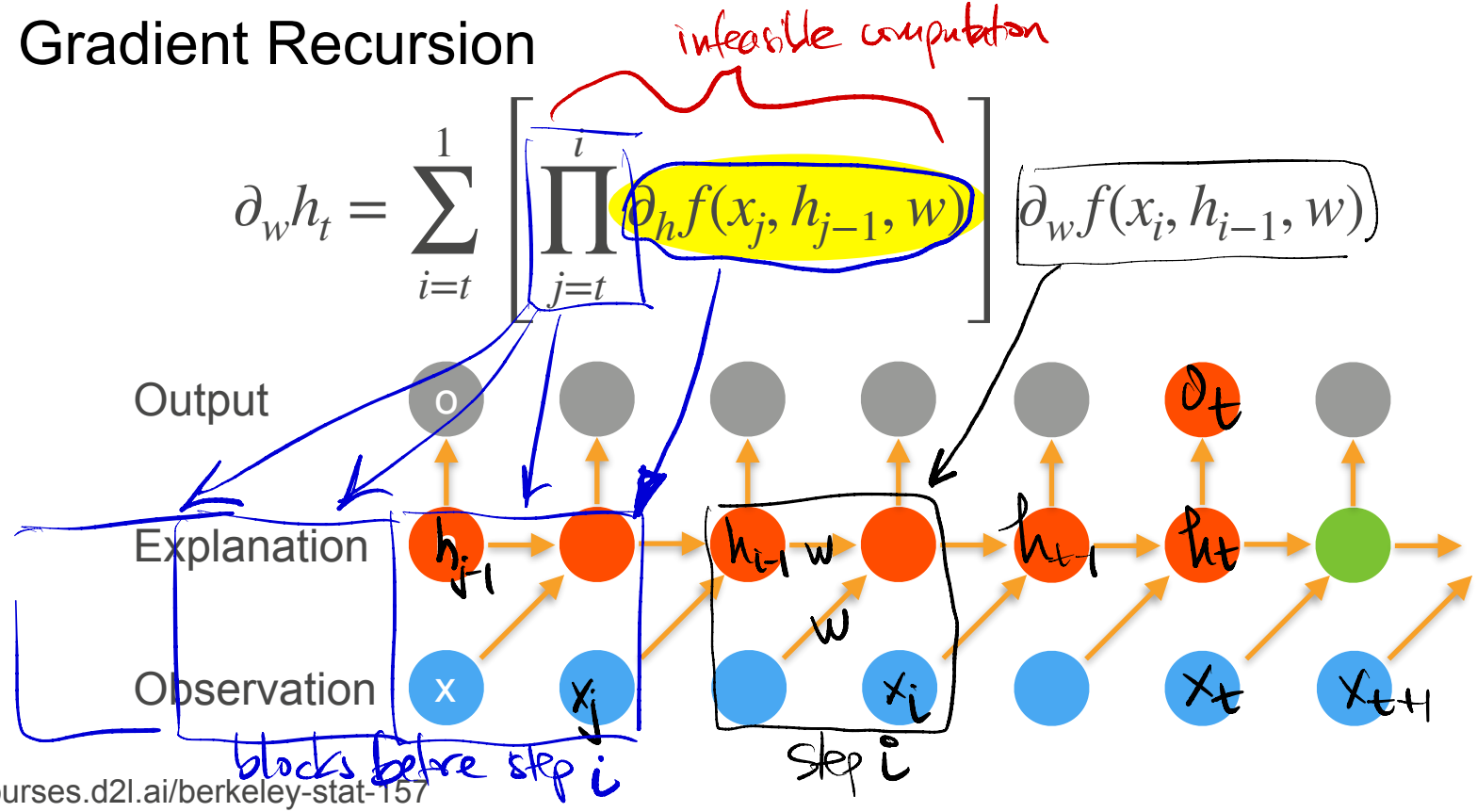
- Gradient Recursion



Latent State Gradient $\partial_w h_t$

- Gradient Recursion

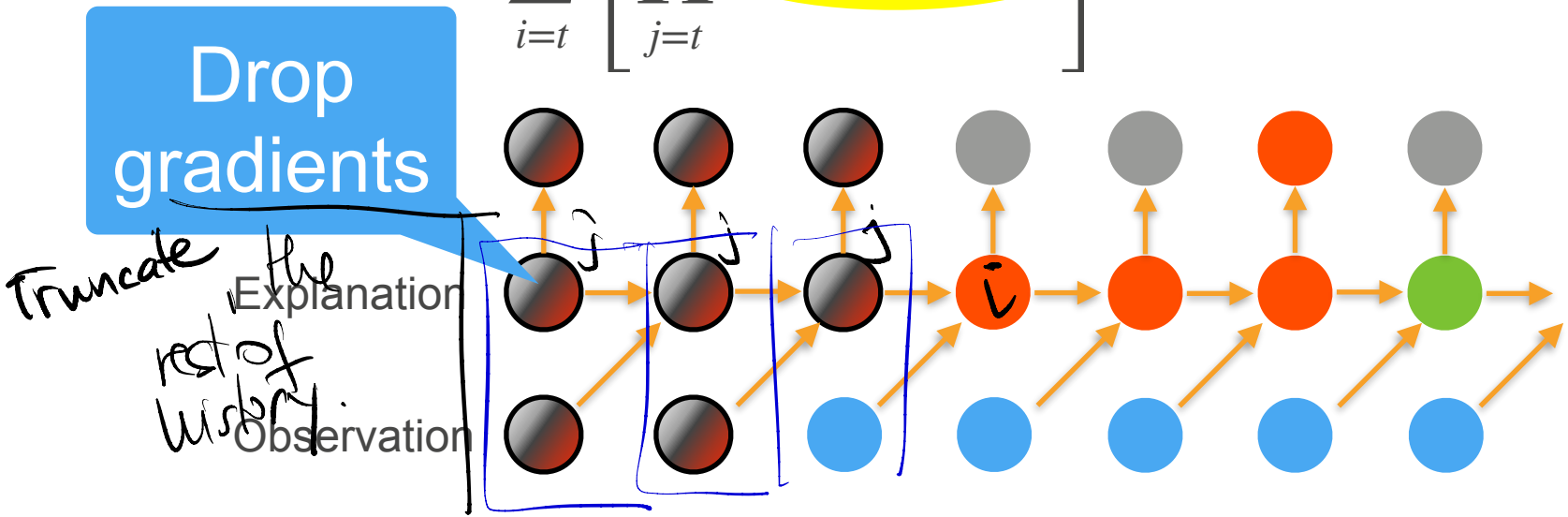
$$\partial_w h_t = \sum_{i=t}^{1} \left[\prod_{j=t}^{i} \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$



Latent State Gradient $\partial_w h_t$

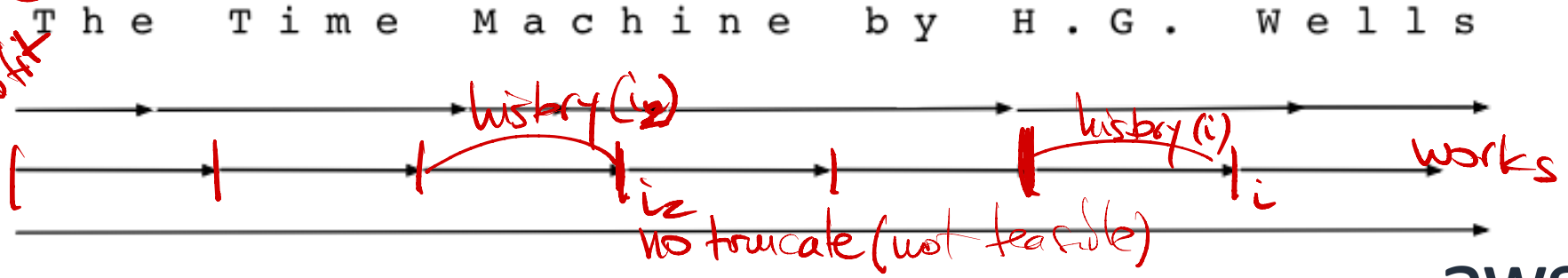
- Gradient Recursion

$$\partial_w h_t = \sum_{i=t}^{1} \left[\prod_{j=t}^{i} \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$



Truncated BPTT

- Don't truncate (naive strategy, costly and divergent)
- Truncate at fixed intervals
  (standard approach, is approximation but works well)
- Variable length (Tallec and Olivier, 2015)
  (is exact after reweighting, doesn't work better in practice)

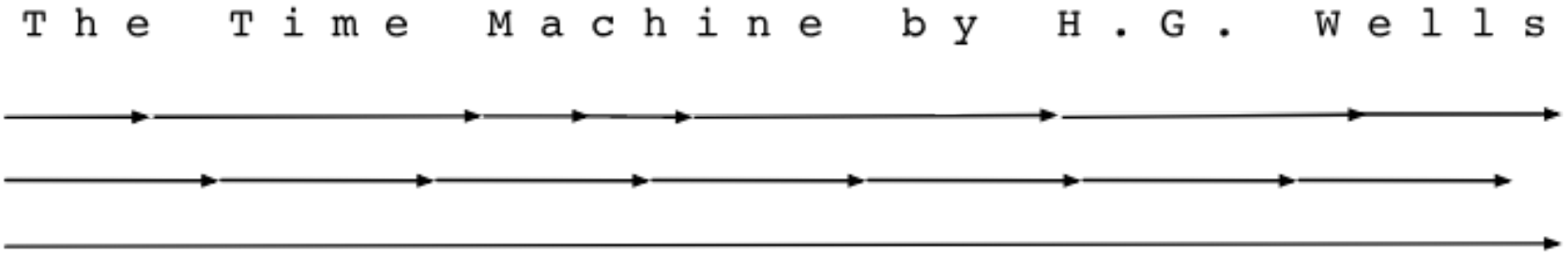


Truncated BPTT

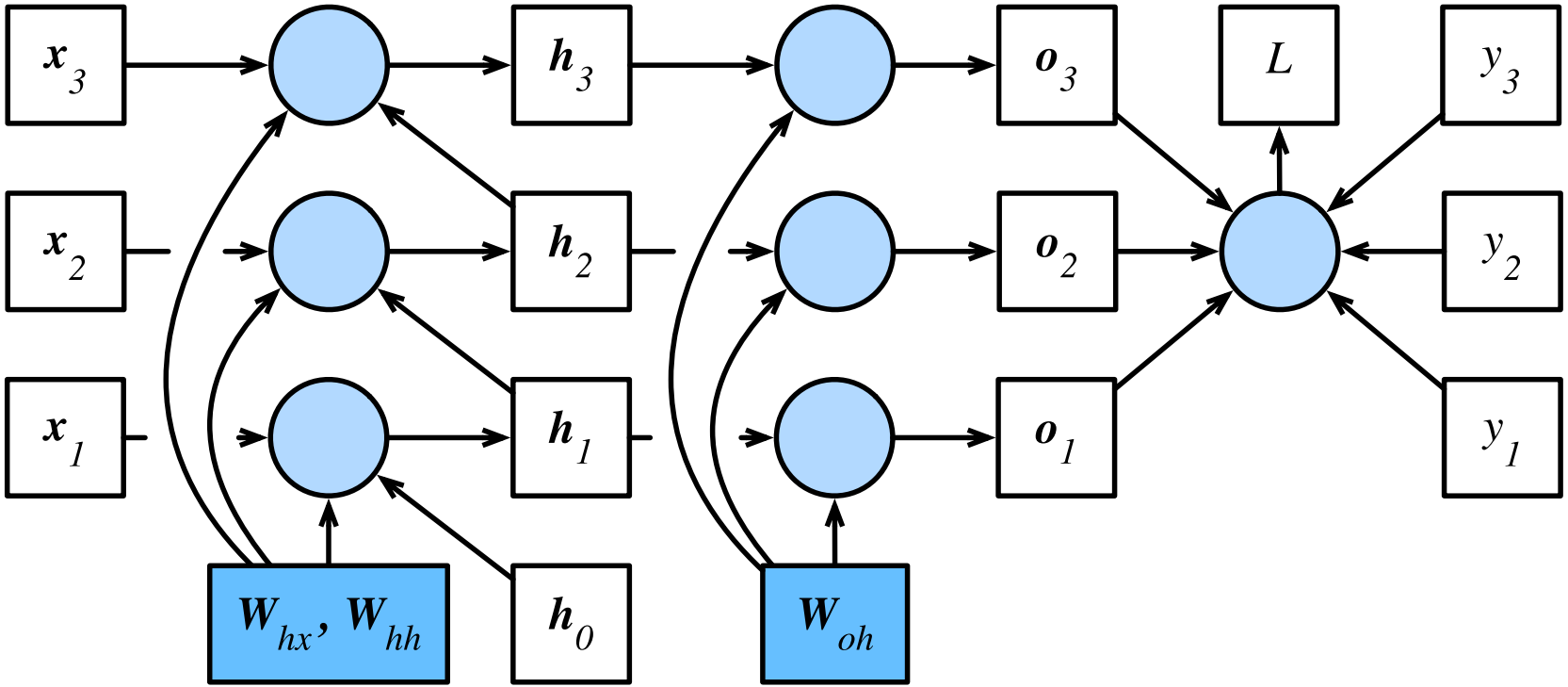
- Random variable instead of simple truncation

$$z_t = \partial_w f(x_t, h_{t-1}, w) + \xi_t \partial_h f(x_t, h_{t-1}, w) \partial_w h_{t-1}$$

- Variable length (Tallec and Olivier, 2015)
  (is exact after reweighting, doesn't work better in practice)



Computational Graph



Example in detail

Toy Model

- Linear RNN

$$\mathbf{h}_t = \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} \text{ and } \mathbf{o}_t = \mathbf{W}_{oh}\mathbf{h}_t$$

- Output gradient

$$\partial_{\mathbf{W}_{oh}} L = \sum_{t=1}^{T} \text{prod}\left(\partial_{\mathbf{o}_t} l(\mathbf{o}_t, y_t), \mathbf{h}_t\right)$$

- State update gradient

$$\partial_{\mathbf{W}_{hh}} L = \sum_{t=1}^{T} \text{prod}\left(\partial_{\mathbf{o}_t} l(\mathbf{o}_t, y_t), \mathbf{W}_{oh}, \partial_{\mathbf{W}_{hh}} \mathbf{h}_t\right)$$

$$\partial_{\mathbf{W}_{hx}} L = \sum_{t=1}^{T} \text{prod}\left(\partial_{\mathbf{o}_t} l(\mathbf{o}_t, y_t), \mathbf{W}_{oh}, \partial_{\mathbf{W}_{hx}} \mathbf{h}_t\right)$$

Gradients … continued

- Linear RNN

$$\mathbf{h}_t = \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} \text{ and } \mathbf{o}_t = \mathbf{W}_{oh}\mathbf{h}_t$$

- Recursive update

$$\partial_{\mathbf{h}_t}\mathbf{h}_{t+1} = \mathbf{W}_{hh}^\top \text{ and thus } \partial_{\mathbf{h}_t}\mathbf{h}_T = \left(\mathbf{W}_{hh}^\top\right)^{T-t}$$

- Full recursion

$$\partial_{\mathbf{W}_{hh}}\mathbf{h}_t = \sum_{j=1}^{t}\left(\mathbf{W}_{hh}^\top\right)^{t-j}\mathbf{h}_j$$

$$\partial_{\mathbf{W}_{hx}}\mathbf{h}_t = \sum_{j=1}^{t}\left(\mathbf{W}_{hh}^\top\right)^{t-j}\mathbf{x}_j .$$

Drop gradients

Truncation in practice

- Compute forward pass **across** truncation boundaries
- Backprop only until truncation boundary (typically mini batch boundary, too)
- In code

```
for s in state:
    s.detach()
```

- Good reason for why sequential sampling is much more accurate than random - state is carried through.

lecture 8/1 : HW6, RNN implement for machine translation

- GRU & LSTM
- Encoder – Decoder
- HW6 demo P1, P2

using "gates" to manage sequence

Sequence-2-sequence for translation

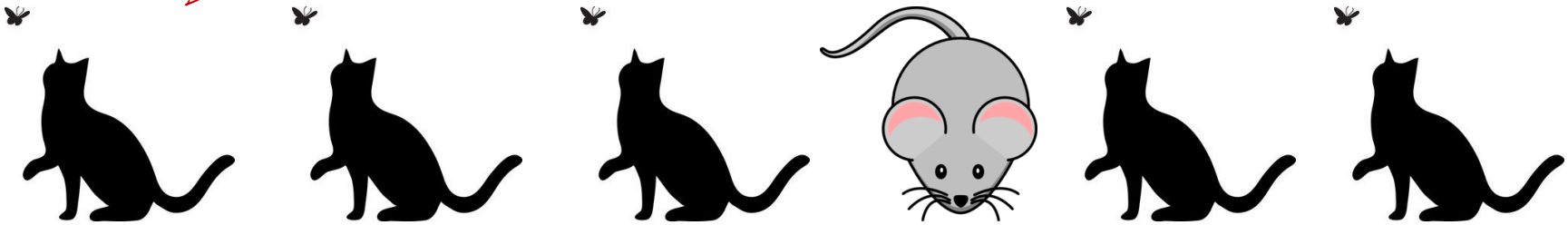
Gated Recurrent Unit (GRU)

- simpler
- faster than LSTM

aws

Paying attention to a sequence

- Not all observations are equally relevant



- Only remember the relevant ones
  - Need mechanism to **pay attention (update gate)**
  - Need mechanism to **forget (reset gate)**

Gating for H_t

$$\boldsymbol{R}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xr} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hr} + \boldsymbol{b}_r),$$
$$\boldsymbol{Z}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xz} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hz} + \boldsymbol{b}_z)$$

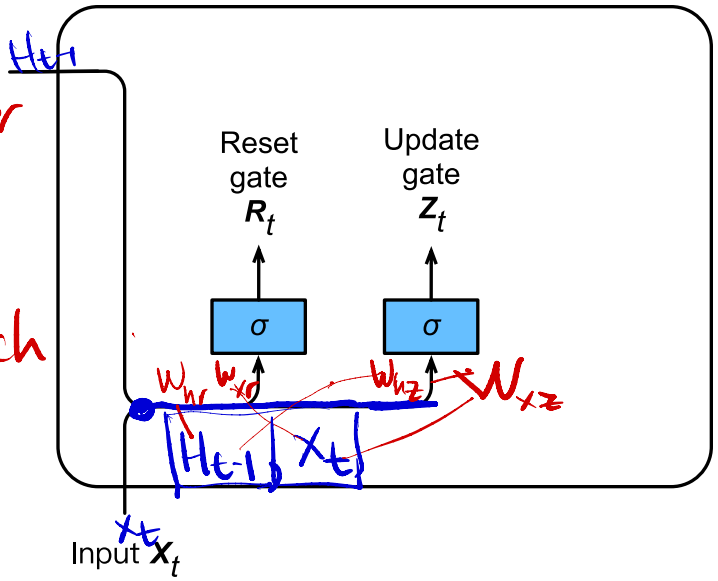
Reset

update

$W_{hr} \neq W_{hz}$

Reset gate : how much of prev H_{t-1} to remember

Update gate : how much duplicate H_{t-1}



Computation matrix

$H_{t-1} \times W$

W to be learned

σ = sigmoid / logistic

σ FC layer with activation fuction

× Element-wise Operator

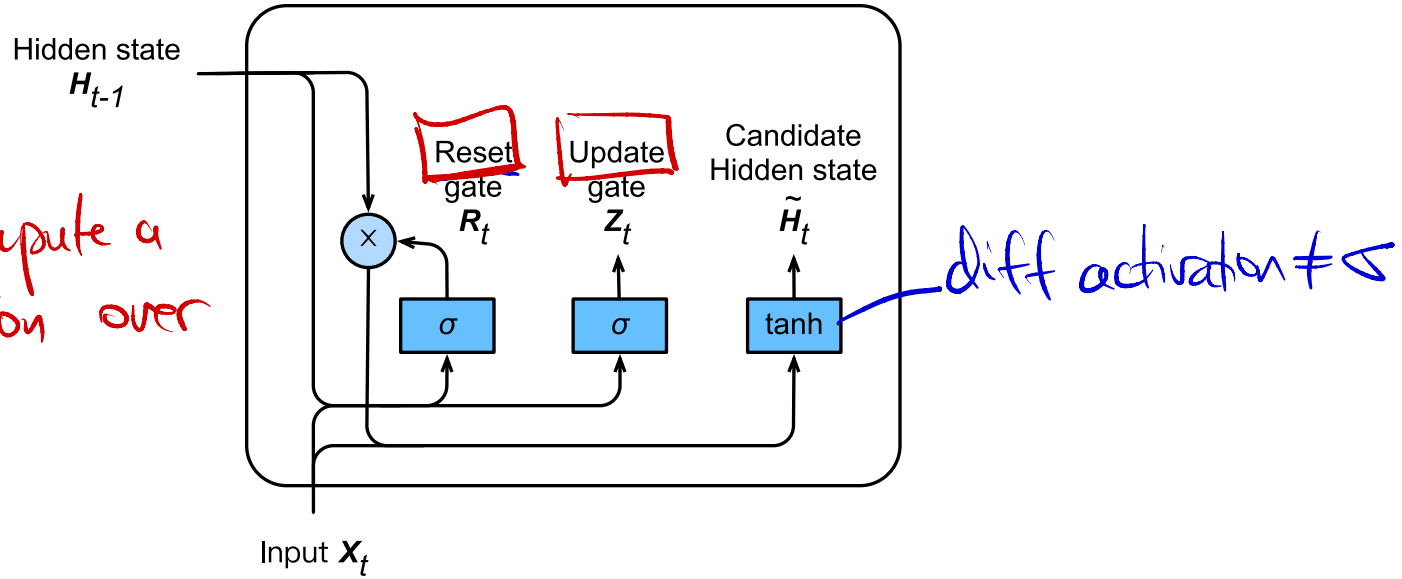
Copy

Concatenate

Candidate Hidden State

$$\tilde{\boldsymbol{H}}_t = \tanh(\boldsymbol{X}_t \boldsymbol{W}_{xh} + (\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1}) \boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

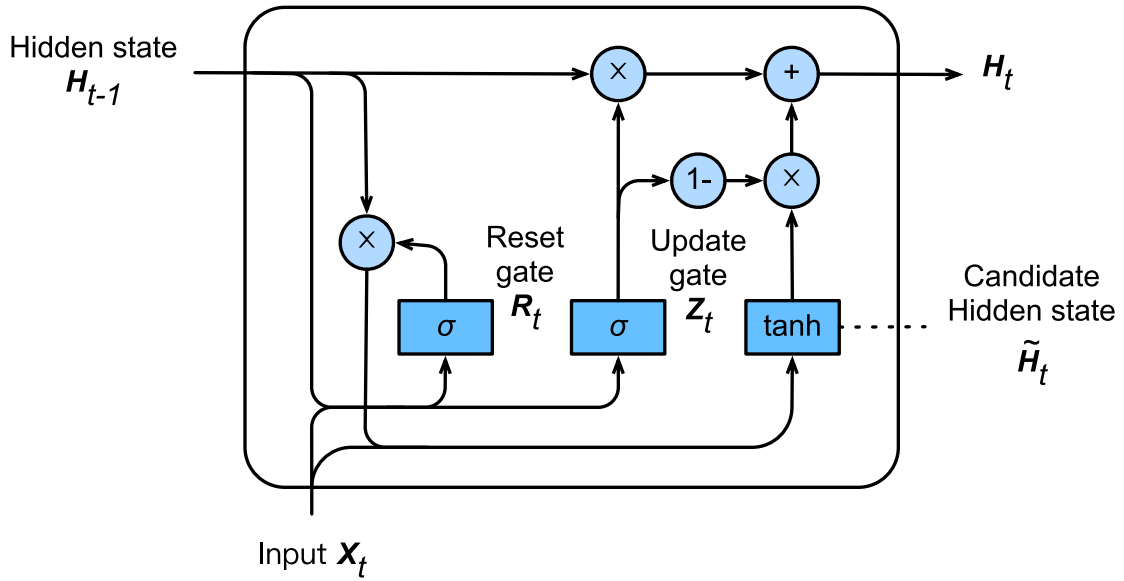
"candidate": We actually compute a prob. distribution over all candidates



Hidden State

retain prev H_{t-1} new calculation

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$



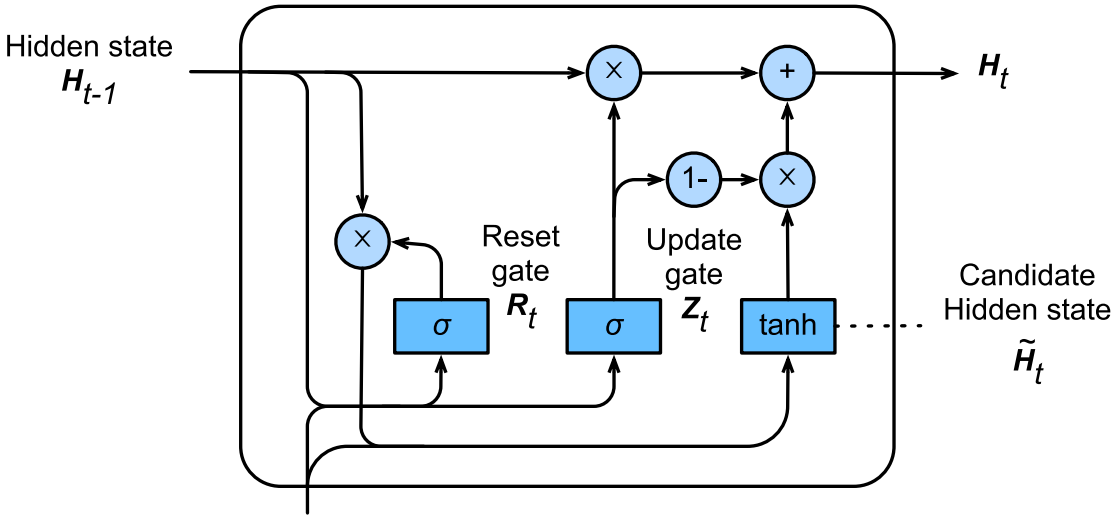
Summary

$$\boldsymbol{R}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xr} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hr} + \boldsymbol{b}_r),$$

$$\boldsymbol{Z}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xz} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hz} + \boldsymbol{b}_z)$$

$$\tilde{\boldsymbol{H}}_t = \tanh(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \left(\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1}\right) \boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

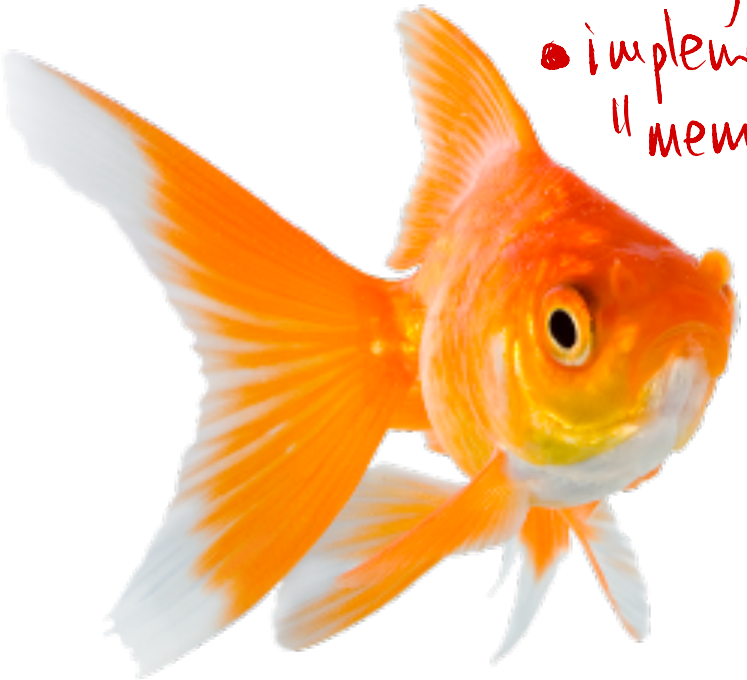
$$\boldsymbol{H}_t = \boldsymbol{Z}_t \odot \boldsymbol{H}_{t-1} + (1 - \boldsymbol{Z}_t) \odot \tilde{\boldsymbol{H}}_t$$



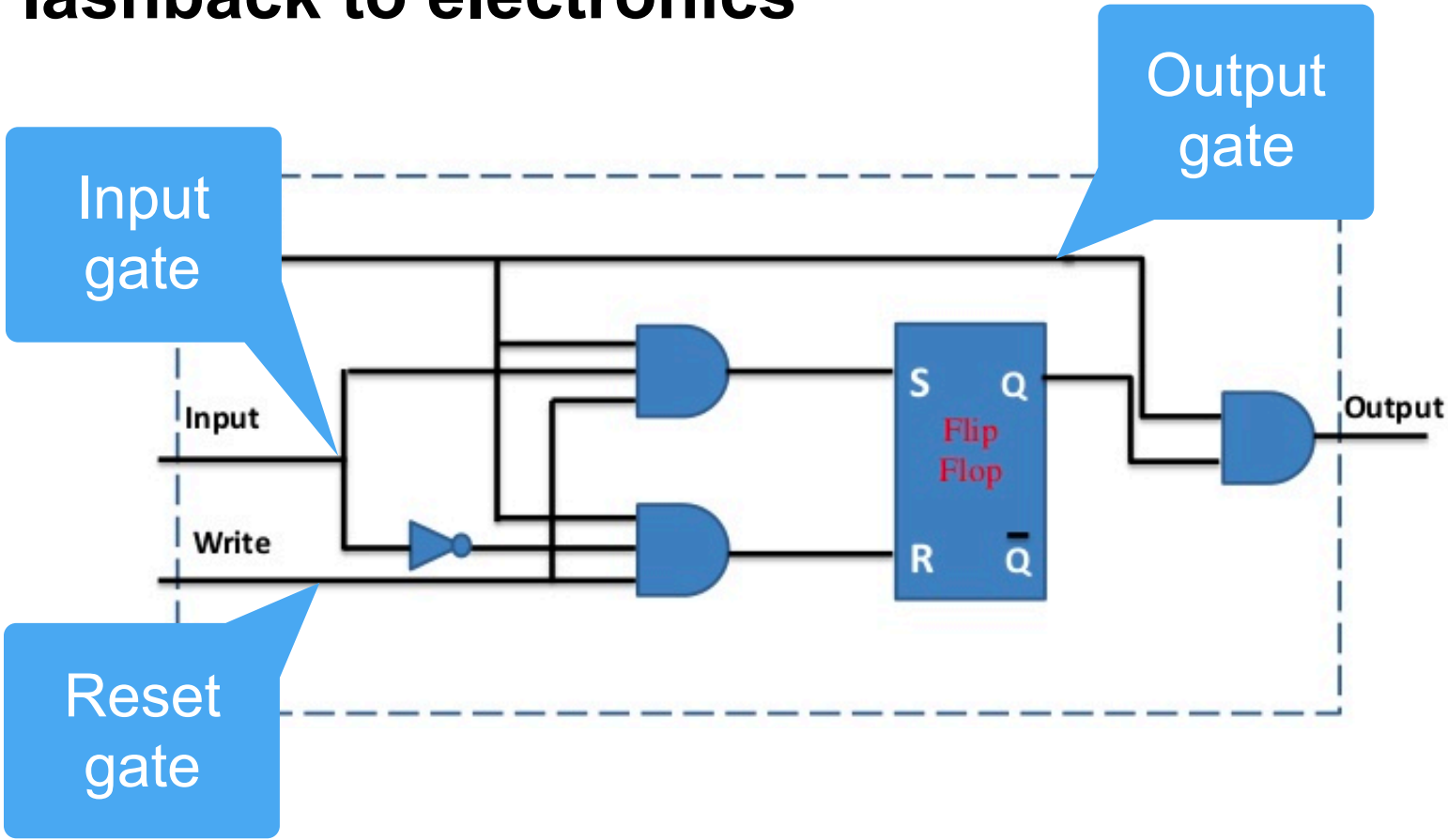
How to manage sequence (steps close "short term"
steps far "long term")

Long Short Term Memory

- using gates
- implement "memory cell"



Flashback to electronics



Long Short Term Memory

- **Forget gate**
  Shrink values towards zero
- **Input gate**
  Decide whether we should ignore the input data
- **Output gate**
  Decide whether the hidden state is used for the output generated by the LSTM

- **Hidden state** and **Memory cell**

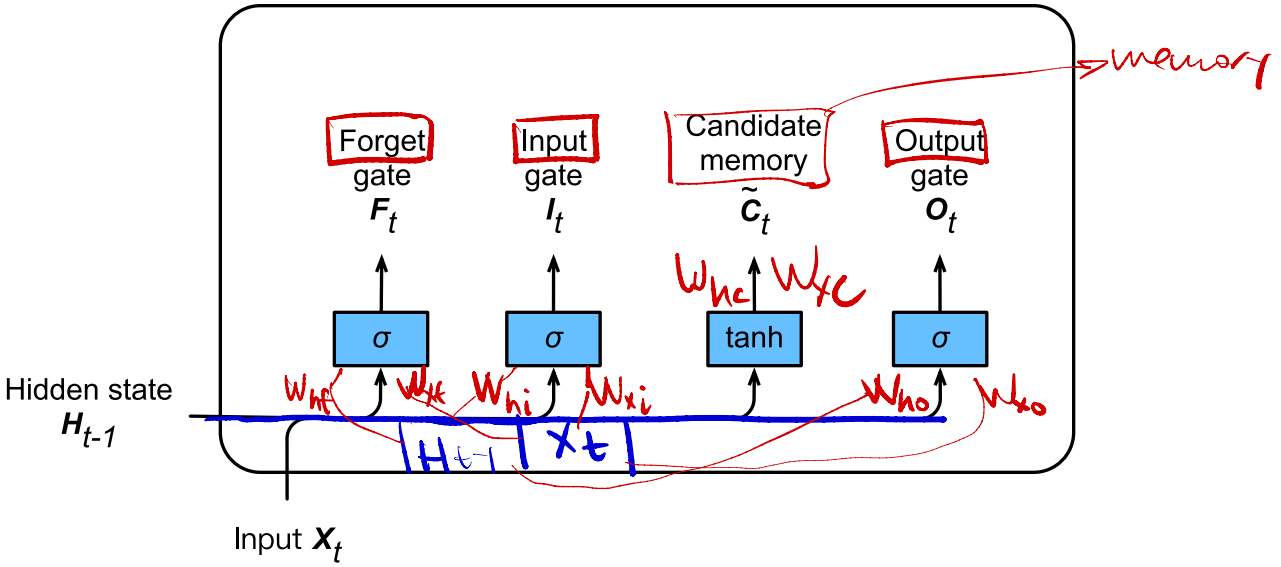
Gates

all calculations similar to GRU gates, each has its own W matrix

$$\boldsymbol{I}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i)$$

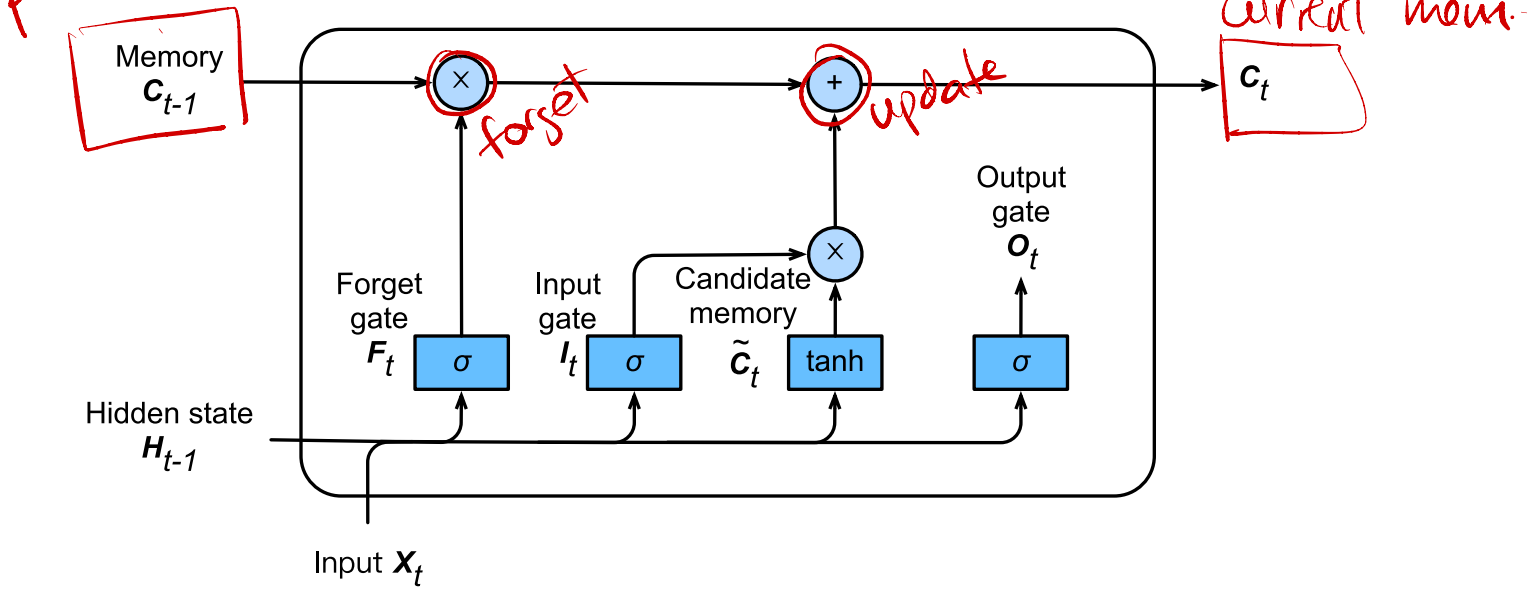
$$\boldsymbol{F}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f)$$

$$\boldsymbol{O}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o)$$



Candidate Memory Cell

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$



σ FC layer with activation fuction

× Element-wise Operator

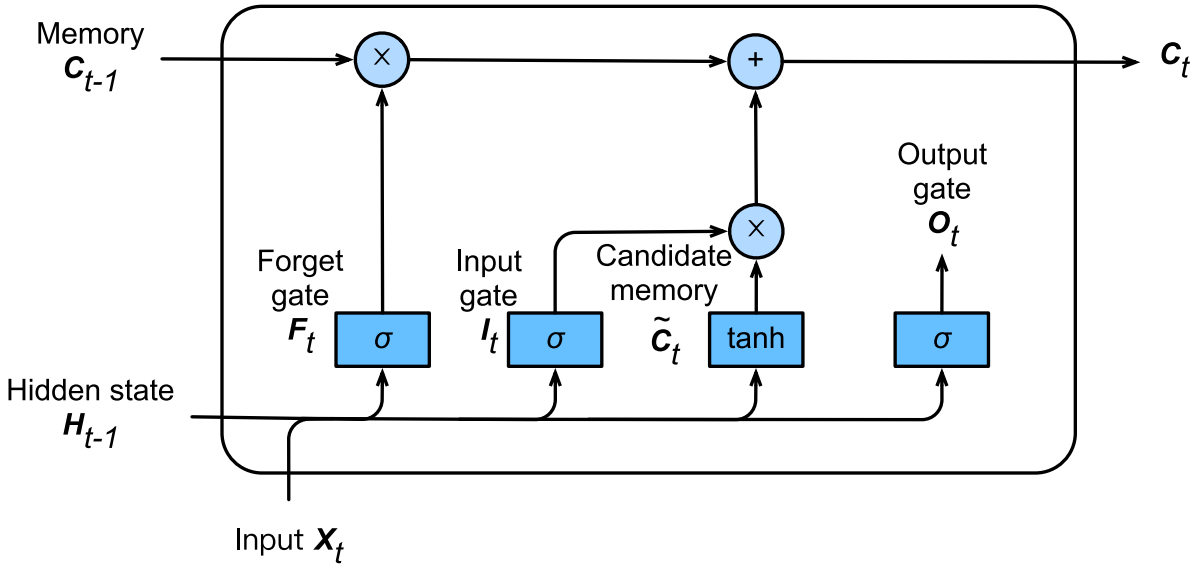
Copy

Concatenate

Memory Cell

element wise prod operator
"Hadamard"

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$



σ FC layer with activation fuction

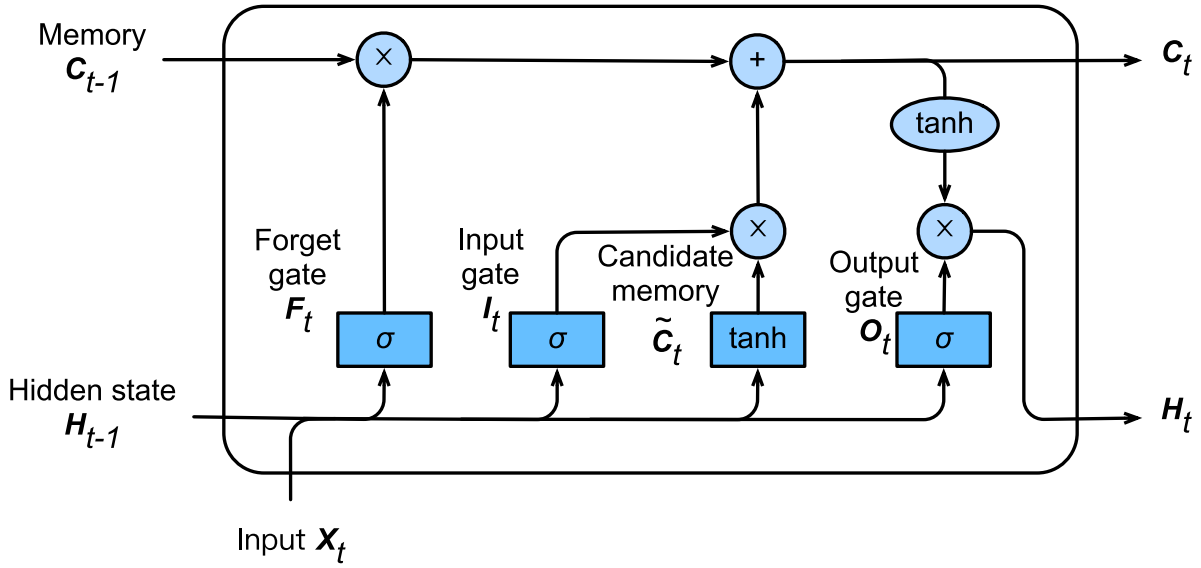
Element-wise Operator

Copy

Concatenate

Hidden State / Output

$$\boldsymbol{H}_t = \boldsymbol{O}_t \odot \tanh(\boldsymbol{C}_t)$$



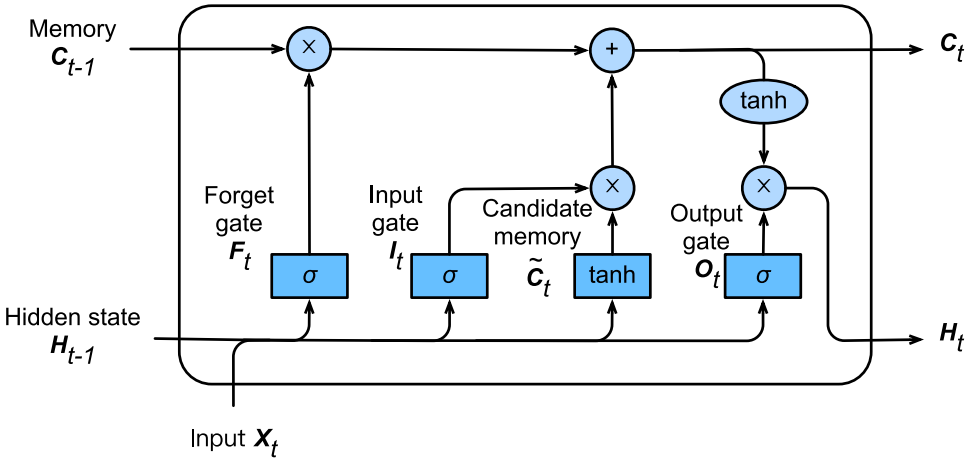
σ FC layer with activation fuction

Element-wise Operator

Copy

Concatenate

Hidden State / Output



$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

$$H_t = O_t \odot \tanh(C_t)$$