Dive into Deep Learning

Chapter 8. Recurrent Neural Networks

Wayne L, Nov 15, 2020

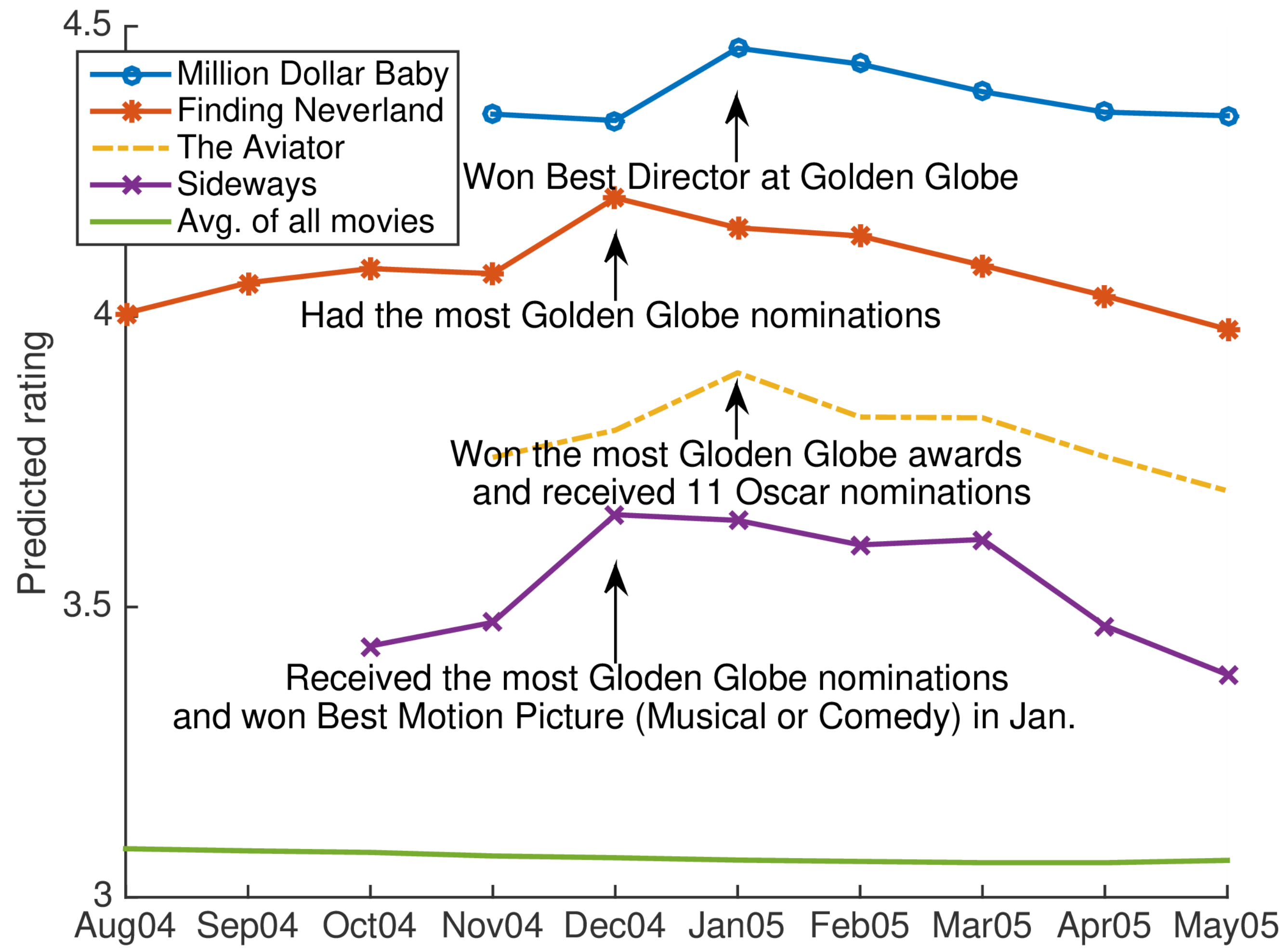
Sequence models

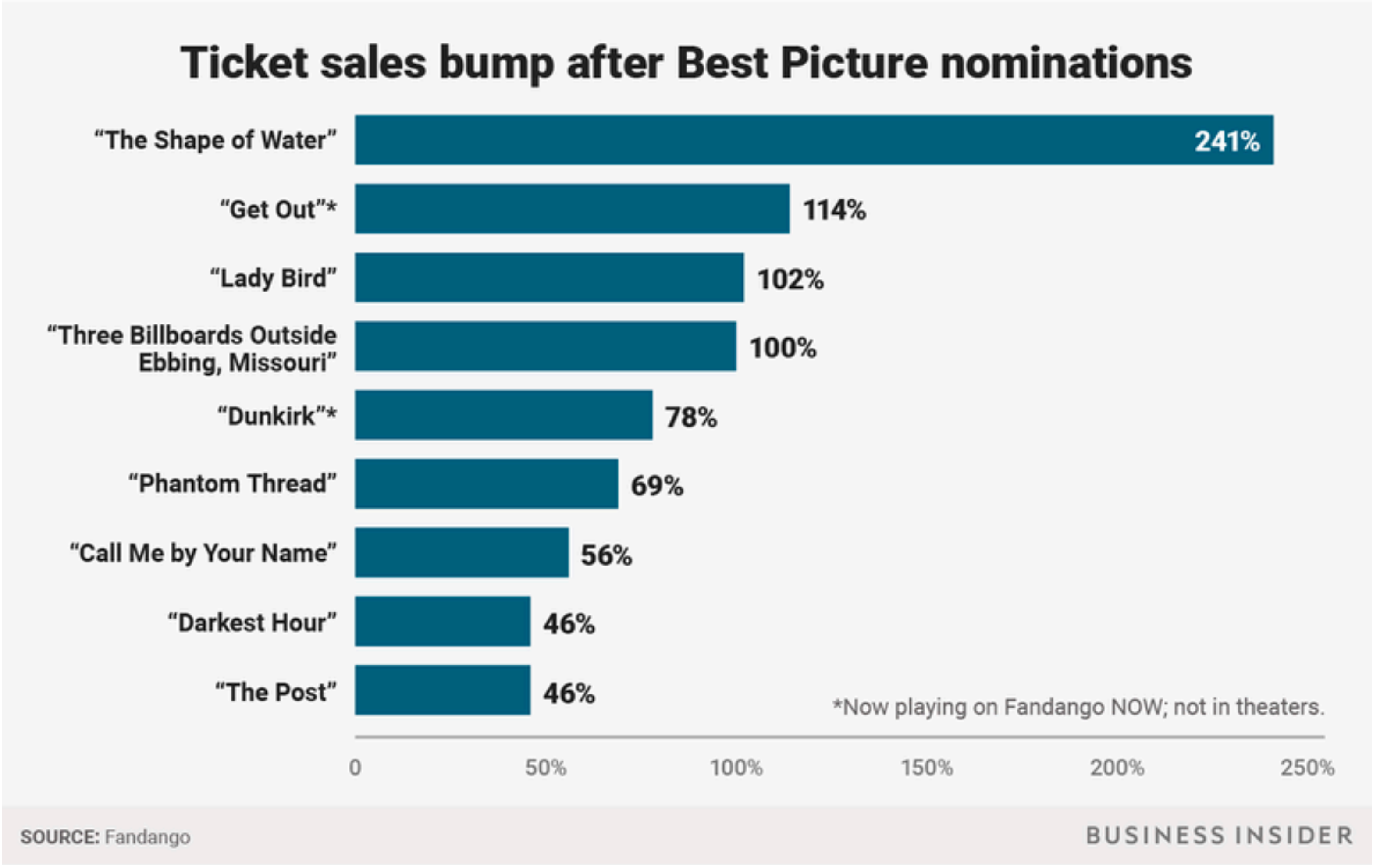
Oscar bump

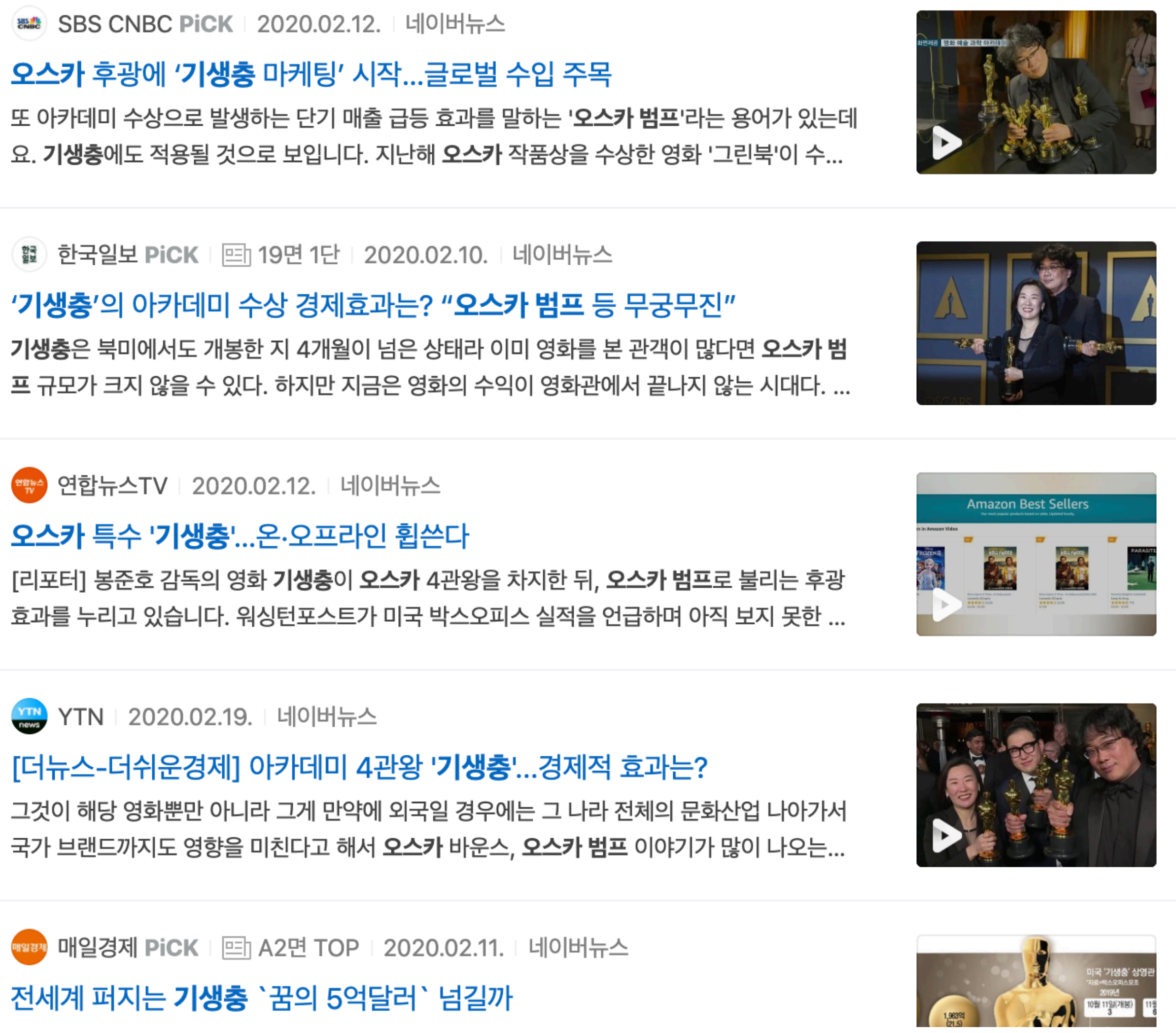
- After the Oscar awards, ratings for the corresponding movie go up, even though it is still the same movie.

- This effect persists for a few months until the award is forgotten. It has been shown that the effect lifts rating by over half a point.

Wu, C.-Y., Ahmed, A., Beutel, A., Smola, A. J., & Jing, H. (2017). Recurrent recommender networks. *Proceedings of the tenth ACM international conference on web search and data mining* (pp. 495–503).



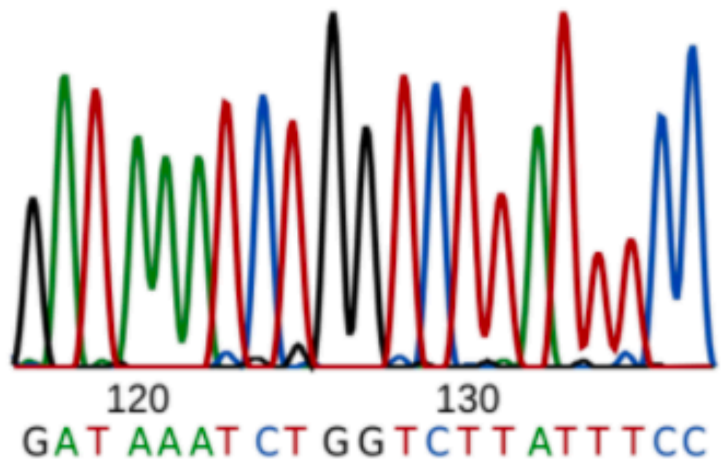




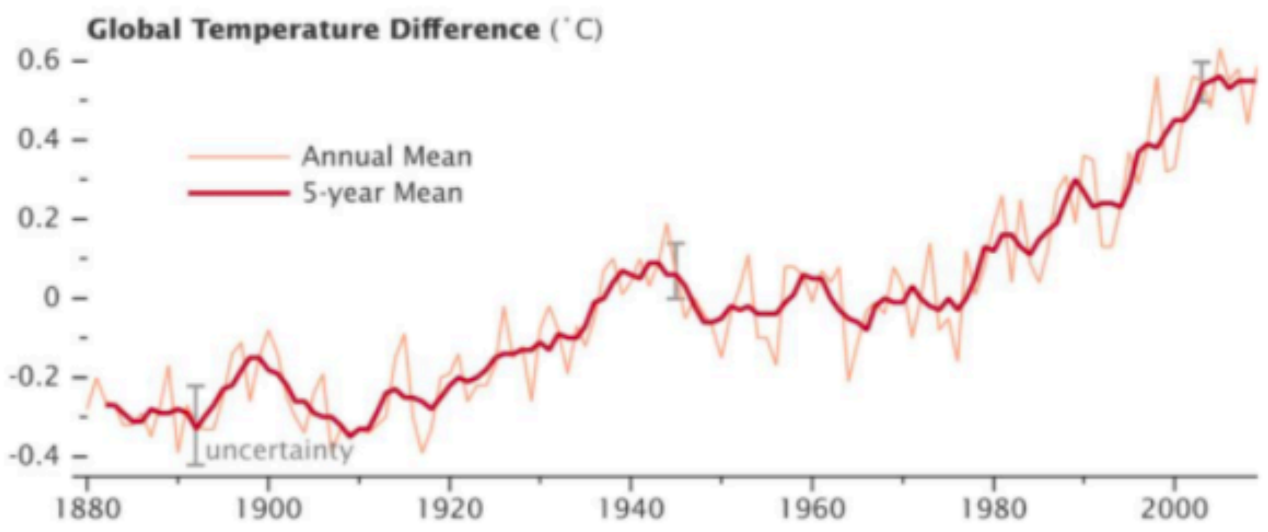
Sequence models

Data usually didn't IID

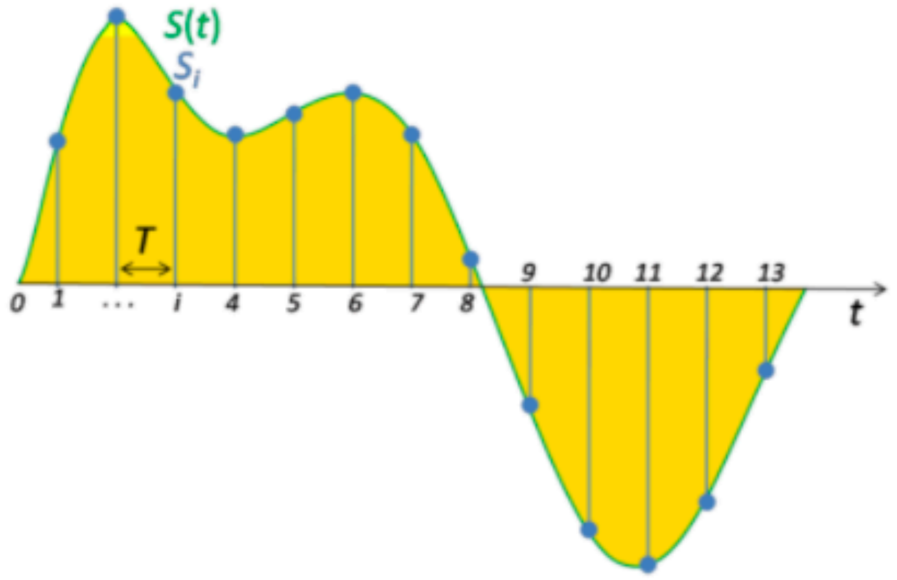
- Various data we have
    - Speech recognition
    - Sentiment classification
    - DNA sequence analysis
    - Machine translation
    - Action recognition
    - NER
    - …



DNA sequence

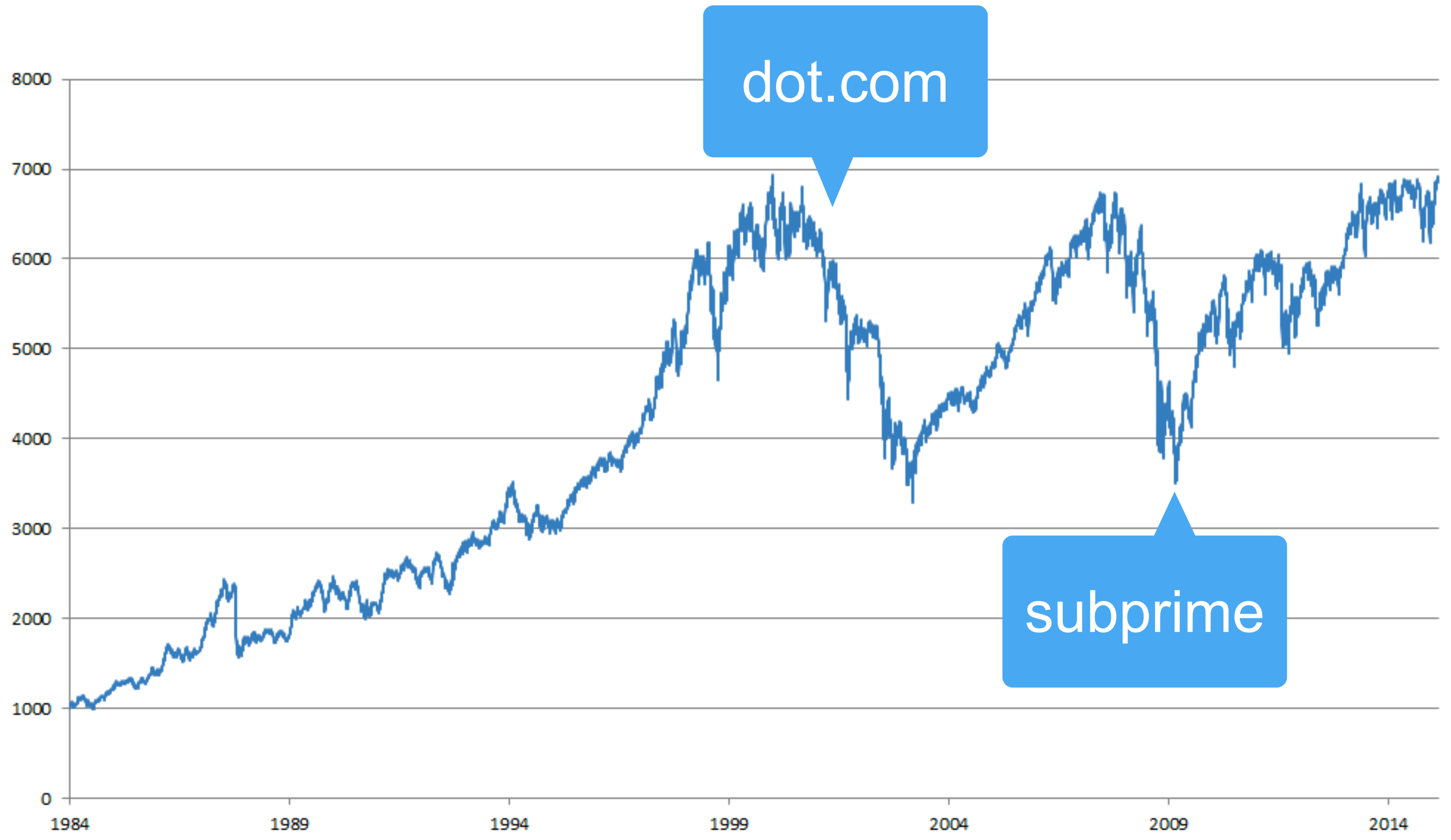


Temperature sequence



Audio sequence

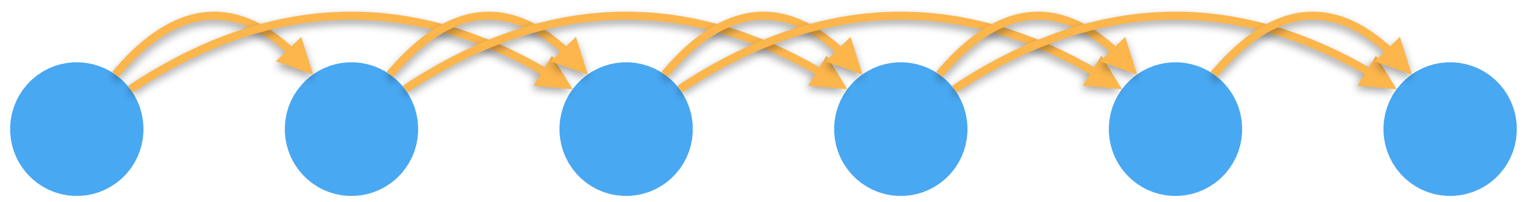
- Suppose that a trader who want to do well in the stock market on day t predicts x_t via $x_t \sim P(x_t \mid x_{t-1}, \ldots, x_1)$.
  (t: time, x_t: price at t)



Autoregressive Models

- Autoregressive model
  - inputs: $x_{t-1}, \ldots, x_1$
  - how to estimate $P(x_t \mid x_{t-1}, \ldots, x_1)$ efficiently
- First, long sequence is not really necessary, instead, only use $x_{t-1}, \ldots, x_{t-\tau}$ $(t > \tau)$
  - # of args always the same, allowing us to train a deep network
  - Such models will be called ***autoregressive model***
- Secondly, to keep some summary h_t of the past observations, and date the same time update h_t in addition to the prediction $\hat{x}_t$ (=$P(x_t \mid h_t)$)
  - These models are also called ***latent auto-regressive models***

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



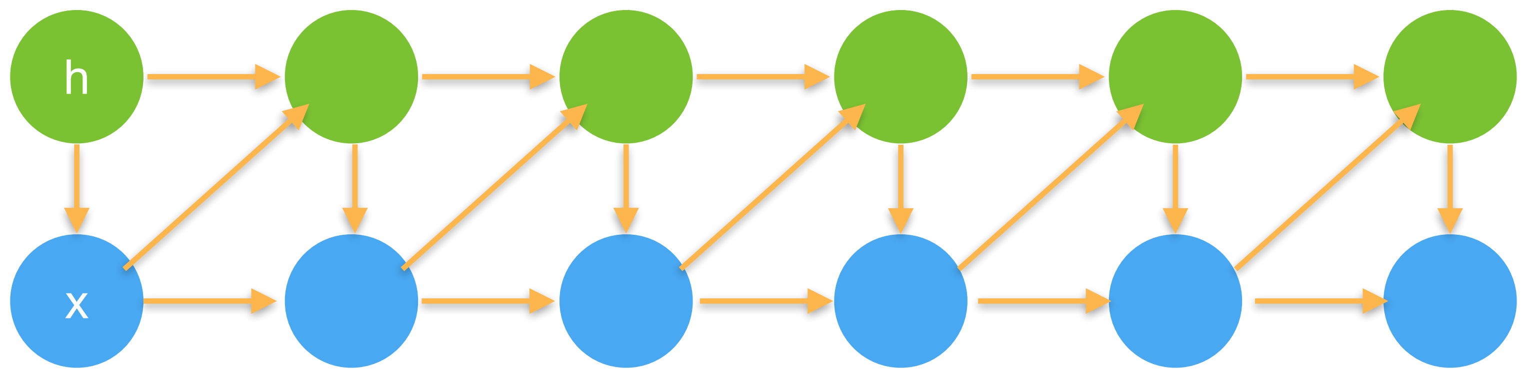
- In practice solve regression problem

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

e.g. train an MLP on previously seen data

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



Markov models

Next observation only depends on the past few terms

- First-order Markov model (if $\tau = 1$)

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{t-1}) \text{ where } P(x_1 \mid x_0) = P(x_1)$$

- Such models are particularly nice whenever x_t assumes **<u>only a discrete value</u>**, since in this case dynamic programming can be used to compute values along the chain exactly.
  For instance, we can compute $P(x_{t+1} \mid x_{t-1})$ efficiently:

$$\begin{aligned} P(x_{t+1} \mid x_{t-1}) &= \frac{\sum_{x_t} P(x_{t+1}, x_t, x_{t-1})}{P(x_{t-1})} \\ &= \frac{\sum_{x_t} P(x_{t+1} \mid x_t, x_{t-1}) P(x_t, x_{t-1})}{P(x_{t-1})} \\ &= \sum_{x_t} P(x_{t+1} \mid x_t) P(x_t \mid x_{t-1}) \end{aligned}$$

$$\frac{P(x_t \cap x_{t-1})}{P(x_{t-1})} = P(x_t \mid x_{t-1})$$

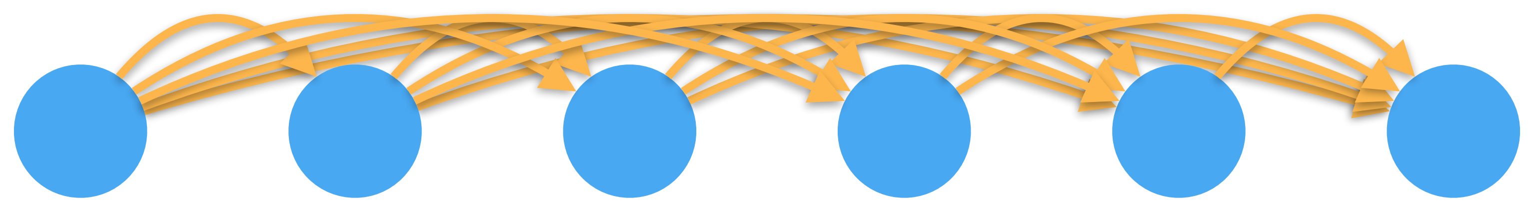
by using the fact that we only need to take into account a very short history of past observations $P(x_{t+1} \mid x_t, x_{t-1}) = P(x_{t+1} \mid x_t)$

Casualty

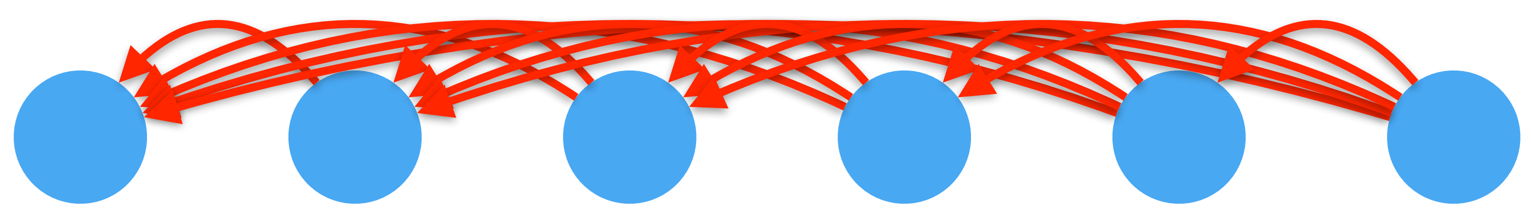
It’s clear that future events cannot influence the past

- Causality (physics) prevents the reverse direction
- ‘wrong’ direction often much more complex to model
- For instance, it has been shown that in some cases we can find $x_{t+1} = f(x_t) + \epsilon$ for some additive noise ϵ, whereas the converse is not true

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



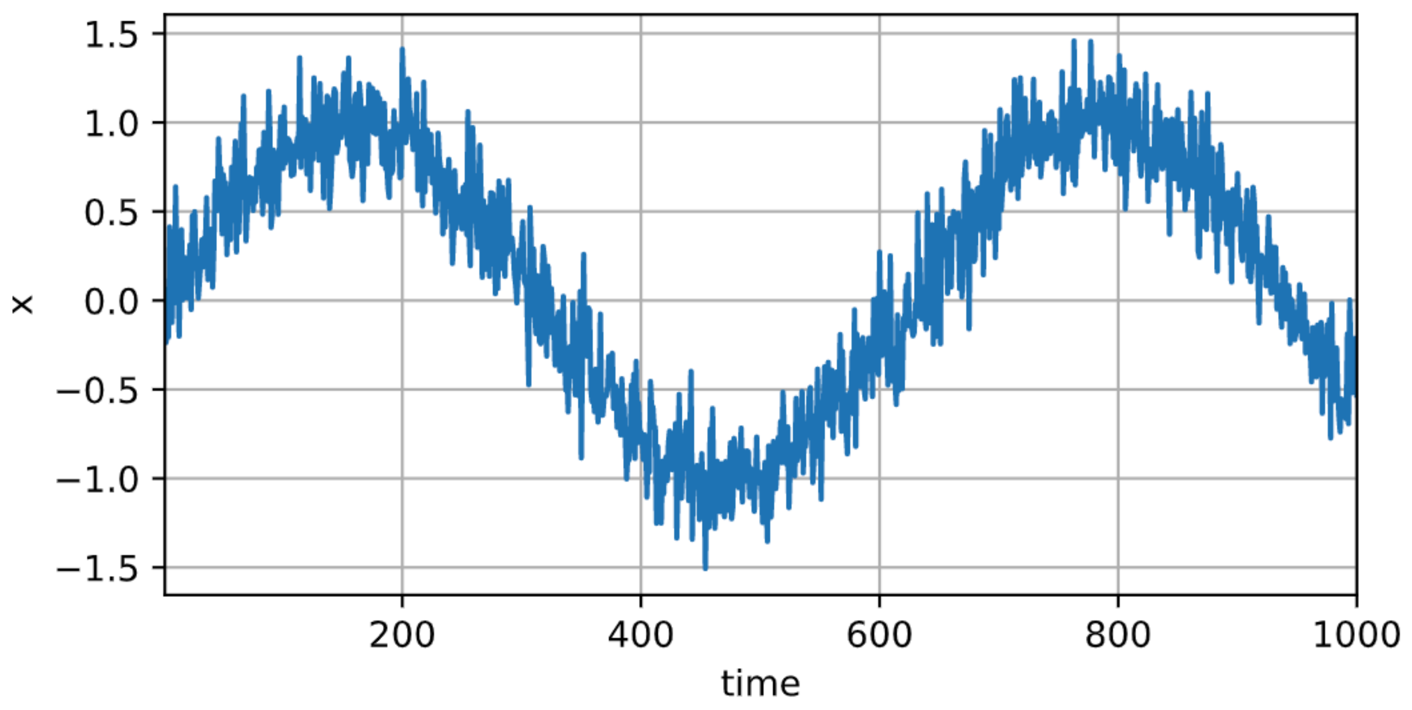
$$p(x) = p(x_T) \cdot p(x_{T-1} \,|\, x_T) \cdot p(x_{T-2} \,|\, x_{T-1}, x_T) \cdot \ldots p(x_1 \,|\, x_2, \ldots x_T)$$



Hoyer, P. O., Janzing, D., Mooij, J. M., Peters, J., & Schölkopf, B. (2009). Nonlinear causal discovery with additive noise models. *Advances in neural information processing systems* (pp. 689–696).

Training sequence model

```
[3] T = 1000  # Generate a total of 1000 points
    time = torch.arange(1, T + 1, dtype=torch.float32)
    x = torch.sin(0.01 * time) + torch.normal(0, 0.2, (T,))
    d2l.plot(time, [x], 'time', 'x', xlim=[1, 1000], figsize=(6, 3))
```



embedding dimension = 4

```
[4] tau = 4
    features = torch.zeros((T - tau, tau))
    for i in range(tau):
        features[:, i] = x[i: T - tau + i]
    labels = d2l.reshape(x[tau:], (-1, 1))
```

```
[5] batch_size, n_train = 16, 600
    # Only the first `n_train` examples are used for training
    train_iter = d2l.load_array((features[:n_train], labels[:n_train]),
                                batch_size, is_train=True)
```

```
[6] # Function for initializing the weights of the network
    def init_weights(m):
        if type(m) == nn.Linear:
            torch.nn.init.xavier_uniform_(m.weight)

    # A simple MLP
    def get_net():
        net = nn.Sequential(nn.Linear(4, 10),
                            nn.ReLU(),
                            nn.Linear(10, 1))
        net.apply(init_weights)
        return net

    # Square loss
    loss = nn.MSELoss()
```

MLP with 2 FC layers, ReLU and squared loss

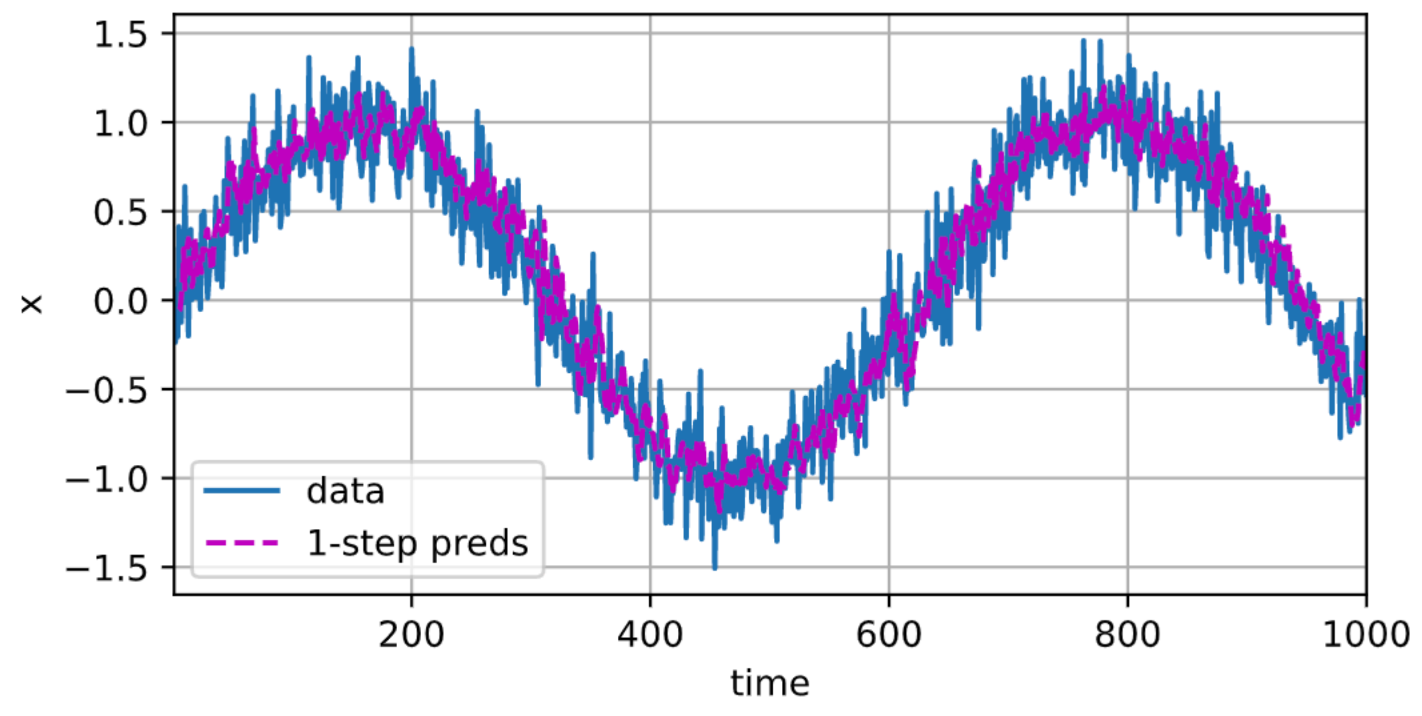
```
[7] def train(net, train_iter, loss, epochs, lr):
        trainer = torch.optim.Adam(net.parameters(), lr)
        for epoch in range(epochs):
            for X, y in train_iter:
                trainer.zero_grad()
                l = loss(net(X), y)
                l.backward()
                trainer.step()
            print(f'epoch {epoch + 1}, '
                  f'loss: {d2l.evaluate_loss(net, train_iter, loss):f}')

    net = get_net()
    train(net, train_iter, loss, 5, 0.01)
```

```
epoch 1, loss: 0.054968
epoch 2, loss: 0.055680
epoch 3, loss: 0.058150
epoch 4, loss: 0.049353
epoch 5, loss: 0.050014
```

Training sequence model

1-step-ahead prediction

```
[8] onestep_preds = net(features)
    d2l.plot([time, time[tau:]], [d2l.numpy(x), d2l.numpy(onestep_preds)], 'time',
             'x', legend=['data', '1-step preds'], xlim=[1, 1000],
             figsize=(6, 3))
```

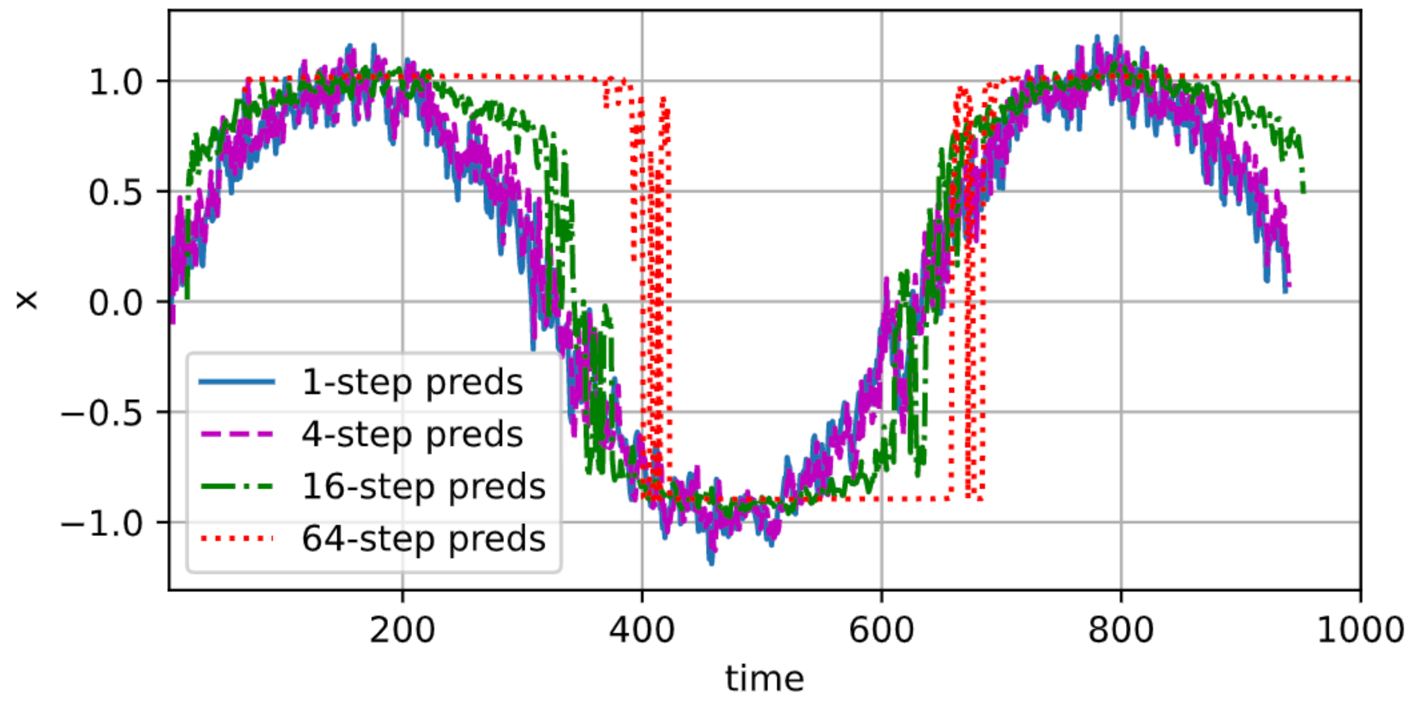


k-step-ahead prediction

```
[11] max_steps = 64
```

```
[12] features = torch.zeros((T - tau - max_steps + 1, tau + max_steps))
     # Column `i` (`i` < `tau`) are observations from `x` for time steps from
     # `i + 1` to `i + T - tau - max_steps + 1`
     for i in range(tau):
         features[:, i] = x[i: i + T - tau - max_steps + 1].T

     # Column `i` (`i` >= `tau`) are the (`i - tau + 1`)-step-ahead predictions for
     # time steps from `i + 1` to `i + T - tau - max_steps + 1`
     for i in range(tau, tau + max_steps):
         features[:, i] = d2l.reshape(net(features[:, i - tau: i]), -1)
```

```
[13] steps = (1, 4, 16, 64)
     d2l.plot([time[tau + i - 1: T - max_steps + i] for i in steps],
              [d2l.numpy(features[:, tau + i - 1]) for i in steps], 'time', 'x',
              legend=[f'{i}-step preds' for i in steps], xlim=[5, 1000],
              figsize=(6, 3))
```



Text Processing

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)

Language Models and the Dataset

- Given $x_1, x_2, \ldots, x_T$ (text sequence of length T), the goal of a language model is to estimate the joint probability of the sequence $P(x_1, x_2, \ldots, x_T)$
- For instance, an ideal language model would be able to generate natural text just on its own, simply by drawing one token at a time $x_t \sim P(x_t \mid x_{t-1}, \ldots, x_1)$.
- LMs are of great service even in their limited form.
- the phrases “to recognize speech” and “to wreck a nice beach” sound very similar. (ambiguity)
- “dog bites man” vs. “man bites dog”
  “I want to eat grandma” vs. “I want to eat, grandma”

Language Models and the Dataset

- Tokenize text data at the word level, applying basic probability rules

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

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

- In order to compute the LM, we need to calculate probability of words and the conditional probability of a word given the previous few words.

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

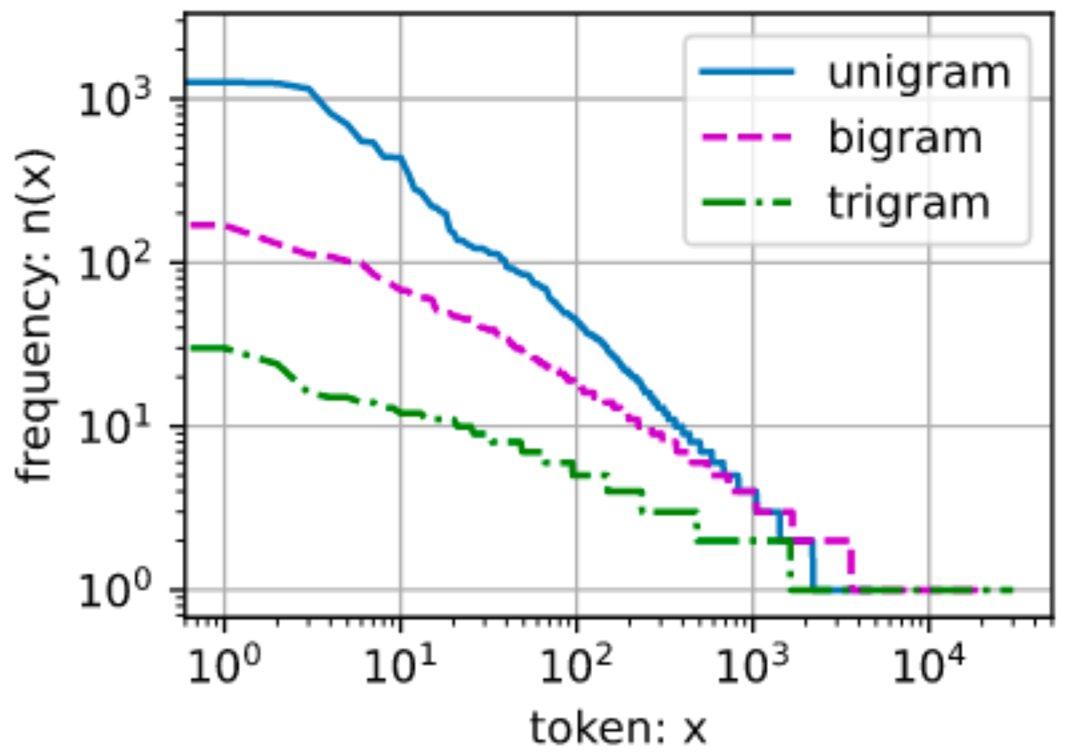
  ($n(x)$ and $n(x, n')$ are the number of occurrences of singletons and consecutive word pairs)

- ***Laplace smoothing***

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

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

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



Zipf's law: $n_i \propto \frac{1}{i^\alpha}$,