Supervised learning: a summary

- A supervised learning paradigm assumes that there are correct labels, sequences of labels, or trees and graphs.
- Having correct labels allows us to compare the predictions of a model with the correct labels to compute the loss of a model.
- During training, we initialize the parameters of a model and use these initial parameters to make predictions. We'll see how wrong these parameters are by computing the loss and update the parameters to reduce this loss.
- When the training is done, we make predictions by searching for the label with the highest score.
- The key to supervised learning is to have annotated data with correct labels. Is there anything we can do without annotated data?

Beyond Supervised Learning

There are other learning scenarios where labeled training sets are available to various degree or not available at all

- When there is no labeled data at all, we'll have to do **unsupervised learning** (e.g., K-Means clustering, variants of the EM algorithm)
- When there is a small amount of labeled data, we might want to try **semi-supervised learning**
- When there is a lot of labeled data in one domain but there is only a small of labeled data in the target domain, we might try **domain adaptation**

K-Means clustering algorithm

K-means clustering algorithm

procedure K-MEANS($\boldsymbol{x}_{1:N}, K$)

  for $i \in 1 \cdots N$ **do**

    $z^{(i)} \leftarrow$ RandomInt$(1, K)$ ▷ Intializing class membership

  repeat

    for $k \in 1 \cdots K$ **do** ▷ recompute cluster centers

      $\boldsymbol{\upsilon}_k \leftarrow \frac{1}{\sum_{i=1}^{N} \delta(z^{(i)}=k)} \sum_{i=1}^{N} \delta(z^{(i)} = k) \boldsymbol{x}^{(i)}$

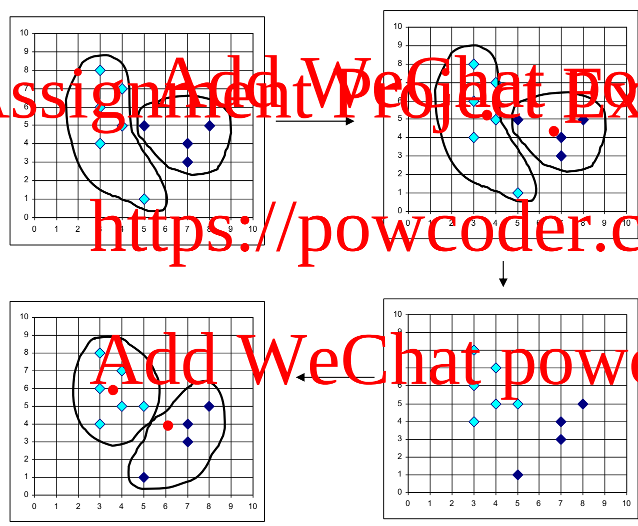
    for $i \in 1 \cdots N$ **do**

      $z^{(i)} \leftarrow \text{argmin}_K \, ||\boldsymbol{x}^{(i)} - \boldsymbol{\upsilon}_k||^2$

  until Converged

  return $z^{(i)}$

K-Means training



- K-means clustering is non-parametric and has no parameters to update
- As a result, there is also no separate training and test phase.
- The number of clusters needs to be pre-specified before the

Semi-supervised learning

- Initialize parameters with supervised learning and then apply unsupervised learning (such as the EM algorithm)

- Multi-view learning: Co-training

  - divide features into multiple views, and train a classifier for each view
  - Each classifier predicts labels for a subset of the unlabeled instances, using only the features available in its view. These predictions are then used as ground truth to train the classifiers associated with the other views

  - Named entity example: named entity view and local context view
  - Word sense disambiguation: local context view and global context view

Multi-view Learning: co-training

	$\boldsymbol{x}^{(1)}$	$\boldsymbol{x}^{(2)}$	y
1.	Peachtree Street	located on	LOC
2.	Dr. Walker	said	PER
3.	Zanzibar	located in	? $\rightarrow$ LOC
4.	Zanzibar	flew to	? $\rightarrow$ LOC
5.	Dr. Robert	recommended	? $\rightarrow$ PER
6.	Oprah	recommended	? $\rightarrow$ PER

Table 5.2: Example of multiview learning for named entity classification

Domain adaptation

Supervised domain adaptation: "Frustratingly simple" domain adaptation (Daumé III, 2007)

- Creates copies of each feature: one for each domain and one for the cross-domain setting

$$\boldsymbol{f}(\boldsymbol{x}, y, d) = \{\text{(boring, NEG, Movie):1, (boring, NEG, *):1,}$$
$$\text{(flavorless, NEG, Movie):1, (flavorless, NEG, *):1,}$$
$$\text{(three-day-old, NEG, Movie):1, (three-day-old, NEG, *):1,}$$
$$\ldots\}$$

where d is the domain.

- Let the learning algorithm allocate weights between domain specific features and cross-domain features: for words that facilitate prediction in both domains, the learner will use cross-domain features. For words that are only relevant to a particular domain, domain-specific features will be used.

Other learning paradigms

- **Active learning:** A learning that is often used to reduce the number of instances that have to be annotated but can still produce the same level of accuracy
- **Distant supervision:** There is no labeled data, but you can generate some (potentially noisy) training data with some external resource such as a dictionary. For example, you can generate named entity annotation with a list of names.
- **Multitask learning:** The learning induces a representation that can be used to solve multiple tasks (learning POS tagging with syntactic parsing)

Expectation Maximization

- An unsupervised iterative learning procedure that has two steps: the Expectation Step and the Maximization Step
- Has many applications in NLP: POS tagging, syntactic parsing, word alignment, clustering
  - The most effective use is in word alignment, which is the first step in statistical machine translation
- Efficient incarnations with dynamic programming:
  - The Forward-Backward algorithm (for tagging)
  - The Inside-Outside algorithm (for parsing)
- The main workhorse for unsupervised learning in NLP

Expectation Maximization for Sequence Labeling

- The naïve EM algorithm for sequence labeling
- A more efficient alternative: the Forward-Backward algorithm

Hidden Markov assumptions

- Recall the generative model for HMM:

$$P(\boldsymbol{x}, \boldsymbol{y}) = P(\boldsymbol{x}|\boldsymbol{y})P(\boldsymbol{y})$$

- Apply the conditional independence assumption that a word only depends on its corresponding tag:

Assignment Project Exam Help
Add WeChat powcoder
https://powcoder.com

$$P(\boldsymbol{x}, \boldsymbol{y}) = \prod_{m=1}^{M} P(x_m|y_m)P(\boldsymbol{y})$$

- Apply the assumption that a tag only depends on its previous tag:

$$P(\boldsymbol{x}, \boldsymbol{y}) = \prod_{m=1}^{M} P(x_m|y_m)P(y_m|y_{m-1})$$

Parameter estimation

- In a supervised setting, the probabilities of the HMM parameters can be computed via a simple count.

- Given two labeled examples:
  - John_N likes_V apples_N
  - John_N likes_V oranges_N

- Counts for the parameters given our usual HMM assumptions:

	N	V	$\blacklozenge$
w_0=John	2	0	0
w_0=likes	0	2	0
w_0=apples	1	0	0
w_0=oranges	1	0	0
$t_{-1} = \Diamond$	2	0	0
t_{-1}=N	0	2	2
t_{-1}=V	2	0	0

Parameter estimation

- Given these counts, we can compute the (unsmoothed) values of the parameters:

	counts			probabilities		
	N	V	♦	N	V	♦
w_0=John	2	0	0	0.5	0	0
w_0=likes	0	2	0	0	1	0
w_0=apples	1	0	0	0.25	0	0
w_0=oranges	1	0	0	0.25	0	0
$t_{-1}=\diamondsuit$	2	0	0	1	0	0
t_{-1}=N	0	2	2	0	0.5	0.5
t_{-1}=V	2	0	0	1	0	0

- We implicitly assume the probabilities of all other labelings are zero:

$$P(\text{John\_V, likes\_N, apples\_V}) = 0$$
$$P(\text{John\_V, likes\_N, apples\_N}) = 0$$
$$\ldots\ldots$$

What if we don't have labeled data?

- Now assume we don't have the labeling, only sequences of word tokens:
  - John likes apples
  - John likes oranges
- We can no longer count the frequencies of bigram tag sequences and how often a tag occurs with a word.
- Instead, let's start with an educated guess of the values of the parameters:

	Expected counts			probabilities		
	N	V	$\blacklozenge$	N	V	$\blacklozenge$
w_0=John	-	-	-	0.5	0.1	0
w_0=likes	-	-	-	0.1	0.5	0
w_0=apples	-	-	-	0.2	0.2	0
w_0=oranges	-	-	-	0.2	0.2	0
$t_{-1} = \lozenge$	-	-	-	0.8	0.2	0
t_{-1}=N	-	-	-	0.1	0.6	0.3
t_{-1}=V	-	-	-	0.6	0.2	0.2

Joint probability of a token sequence and its tag sequence

With the initial assignments, we can compute the joint probability of the token sequence and the tag sequence.

$$P(\text{John\_N, likes\_N, apples\_N}) = P(N|\Diamond)P(N|N)P(N|N)P(\blacklozenge|N)P(John|N)$$
$$P(likes|N)P(apples|N) = 0.8 \times 0.1 \times 0.1 \times 0.3 \times 0.5 \times 0.1 \times 0.2 = 0.000024$$

$P(\text{John\_N, likes\_N, apples\_V}) = 0.000096$

$P(\text{John\_N, likes\_V, apples\_V}) = 0.00096$

$P(\text{John\_N, likes\_V, apples\_N}) = 0.00432$

$P(\text{John\_V, likes\_N, apples\_N}) = 0.0000072$

$P(\text{John\_V, likes\_N, apples\_V}) = 0.000029$

$P(\text{John\_V, likes\_V, apples\_N}) = 0.000072$

$P(\text{John\_V, likes\_V, apples\_V}) = 0.000016$

$P(\text{John\_N, likes\_N, oranges\_N}) = 0.000024$

$P(\text{John\_N, likes\_N, oranges\_V}) = 0.000096$

$P(\text{John\_N, likes\_V, oranges\_V}) = 0.00096$

$P(\text{John\_N, likes\_V, oranges\_N}) = 0.00432$

$P(\text{John\_V, likes\_N, oranges\_N}) = 0.0000072$

$P(\text{John\_V, likes\_N, oranges\_V}) = 0.000029$

$P(\text{John\_V, likes\_V, oranges\_N}) = 0.000072$

$P(\text{John\_V, likes\_V, oranges\_V}) = 0.000016$

Conditional probability of a tag sequence given its token sequence

With the joint probabilities we can compute the conditional probability of a tag sequence given its token sequence

$$P((N, N, N)|(\text{John, likes, apples})) = 0.0043$$
$$P((N, N, V)|(\text{John, likes, apples})) = 0.0174$$
$$P((N, V, V)|(\text{John, likes, apples})) = 0.174$$
$$P((N, V, N)|(\text{John, likes, apples})) = 0.782$$
$$P((V, N, N)|(\text{John, likes, apples})) = 0.0013$$
$$P((V, N, V)|(\text{John, likes, apples})) = 0.0052$$
$$P((V, V, N)|(\text{John, likes, apples})) = 0.013$$
$$P((V, V, V)|(\text{John, likes, apples})) = 0.0029$$
$$P((N, N, N)|(\text{John, likes, oranges})) = 0.0043$$
$$P((N, N, V)|(\text{John, likes, oranges})) = 0.0174$$
$$P((N, V, V)|(\text{John, likes, oranges})) = 0.174$$
$$P((N, V, N)|(\text{John, likes, oranges})) = 0.782$$
$$P((V, N, N)|(\text{John, likes, oranges})) = 0.0013$$
$$P((V, N, V)|(\text{John, likes, oranges})) = 0.0052$$
$$P((V, V, N)|(\text{John, likes, oranges})) = 0.013$$
$$P((V, V, V)|(\text{John, likes, oranges})) = 0.0029$$

Expected counts of the parameters

With the conditional probabilities of each tag sequence given its token sequence, we can compute the expected count of each parameter – the count of each parameter weighted by the conditional probability of the tagged sequence it appears in:

	Expected counts			probabilities		
	N	V	♦	N	V	♦
w_0=John	1.955	0.0448	0	-	-	-
w_0=likes	0.056	?	0	-	-	-
w_0=apples	0.80	?	0	-	-	-
w_0=oranges	0.80	?	?	-	-	-
$t_{-1} = \Diamond$	1.995	0.0448	0	-	-	-
t_{-1}=N	0.0546	1.957	1.601	-	-	-
t_{-1}=V	?	?	?	-	-	-

Maximization

With the expected count of the parameters we can re-estimate (via maxization) the parameters, and replace the initial parameters with the updated parameters:

	Expected counts			probabilities		
	N	V	♦	N	V	♦
w_0=John	1.956	0.0448	0	0.5414	-	-
w_0=likes	0.056	?	0	-	-	-
w_0=apples	0.80	?	0	-	-	-
w_0=oranges	0.80	?	?	-	-	-
$t_{-1} = \Diamond$	1.995	0.0448	0	0.978	-	-
t_{-1}=N	0.0546	1.957	1.601	-	0.5417	-
t_{-1}=V	?	?	?	-	-	-

With the updated parameters, we can iterate this process...

A summary of this process:

- We first initialize the parameters with some initial values, hopefully with some prior knowledge
- The E-Step:
  - Using these parameters, we can compute the conditional probability of a tag sequence
  - With the conditional probabilities we can compute the expected counts of the parameters
- The M-step: we then re-estimate the parameters by maximizing them.
- We repeat this process until it converges at some local maxima. The underlying model is not concave (the opposite of convex) so there is no guarantee that it will hit global maxima.

The generic EM algorithm

Input: A sample of N points, $x^{(1)}, x^{(2)}, \dots, x^{(N)}$. A model $P(x, y|\boldsymbol{\theta})$ of the following form: $P(x, y|\boldsymbol{\theta}) = \prod_{r=1}^{|\theta|} \boldsymbol{\theta}_r^{Count(x,y,r)}$

Output: $\boldsymbol{\theta}^T$

Initialization: Choose some initial value for $\boldsymbol{\theta}^0$

for $t \leftarrow 1 \cdots T$ **do**
    for $r = 1 \cdots |\boldsymbol{\theta}|$ **do**
        $\mathbb{E}[Count(r)] = 0$

    for $i = 1 \cdots N$ **do**
        For all y, compute $t_y = P(x^i, y|\boldsymbol{\theta}^{t-1})$

        for all y, set $u_y = t_y / \sum_y t_y$
        for all $r = 1 \cdots |\boldsymbol{\theta}|$, set $\mathbb{E}[Count(r)] = \mathbb{E}[Count(r)] + \sum_y u_y Count(x^i, y, r)$

for $r = 1 \cdots |\boldsymbol{\theta}|$ **do**
    $\boldsymbol{\theta}_r^t = \frac{\mathbb{E}[Count(r)]}{Z}$ where Z is a normalization constant

At this time you should have some questions...

- Does this work? Why does this work at all?
- What are the scenarios in which EM can be deployed?
- Can this be done more efficiently?
- Why do we weight the counts with the posterior of the sequence (and not something else)?
- Will the log likelihood monotonically increase after each iteration?
- We'll try to answer some of them...

The Baum-Welch algorithm

- The naive Expectation Maximization algorithm we outlined above works for short sentences in small data sets, but does not scale.
- The Baum Welch algorithm is an efficient alternative that combines EM with the Forward Backward algorithm.
- In the M-step, the HMM parameters are estimated from expected count:

$$Pr(W = i|Y = k) = \phi_{k,i} = \frac{\mathbb{E}[\text{count}(W = i, Y = k)]}{\mathbb{E}[\text{count}(Y = k)]}$$

$$Pr(Y_m = k|Y_{m-1} = k') = \lambda_{k',k} = \frac{\mathbb{E}[\text{count}(Y_m = k, Y_{m-1} = k')]}{\mathbb{E}[\text{count}(Y_{m-1} = k')]}$$

The E-Step: transition counts

- The local scores follow the usual definitions of HMM:

$$s_m(k, k') = \log P_E(w_m | Y_m = k; \phi) + \log P_T(Y_m = k | Y_{m-1} = k'; \lambda)$$

- The expected transition count for a single instance is:

$$\mathbb{E}[\text{count}(Y_m = k, Y_{m-1} = k') | \boldsymbol{w}] = \sum_{m=1}^{M} Pr(Y_m = k, Y_{m-1} = k' | \boldsymbol{w})$$

$$Pr(Y_m = k, Y_{m-1} = k' | \boldsymbol{w}) = \frac{\alpha_{m-1}(k') \times \exp s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)}$$

- The posterior is computed the same way as in the forward-backward computation in CRF, with the only difference being how the local score is computed.

The E-Step: emission counts

- The local scores follow the usual definitions of HMM:

$$s_m(k, k') = \log P_E(w_m | Y_m = k; \phi) + \log P_T(Y_m = k | Y_{m-1} = k'; \lambda)$$

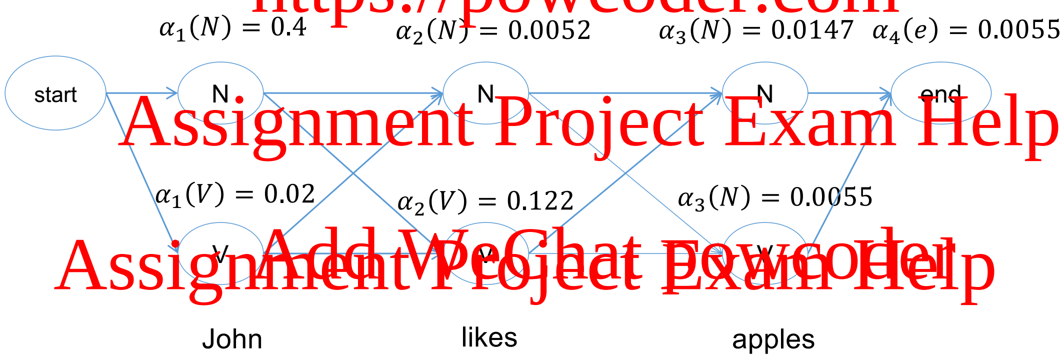
- The expected emission count for a single instance is

$$\mathbb{E}[\text{count}(Y_m = k, w_m) | \boldsymbol{w}] = \sum_{m=1, w_m}^{M} Pr(Y_m = k | \boldsymbol{w})$$

$$\begin{aligned} Pr(Y_m = k | \boldsymbol{w}) &= \sum_{Y_{m-1}=k'} Pr(Y_m = k, Y_{m-1} = k' | \boldsymbol{w}) \\ &= \sum_{Y_{m-1}=k'} \frac{\alpha_{m-1}(k') \times \exp s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)} \\ &= \alpha_m(k) \beta_m(k) \end{aligned}$$

- The posterior is computed the same way as in the forward-backward computation in CRF, with the only difference being how the local score is computed.

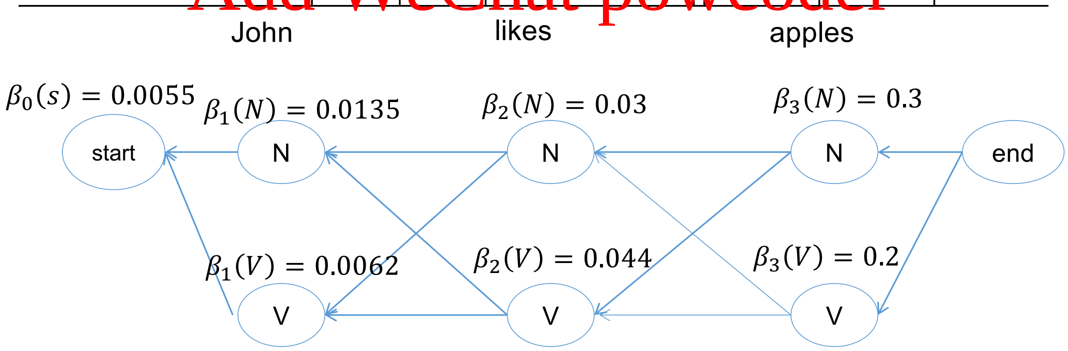
Baum-Welch: Forward computation



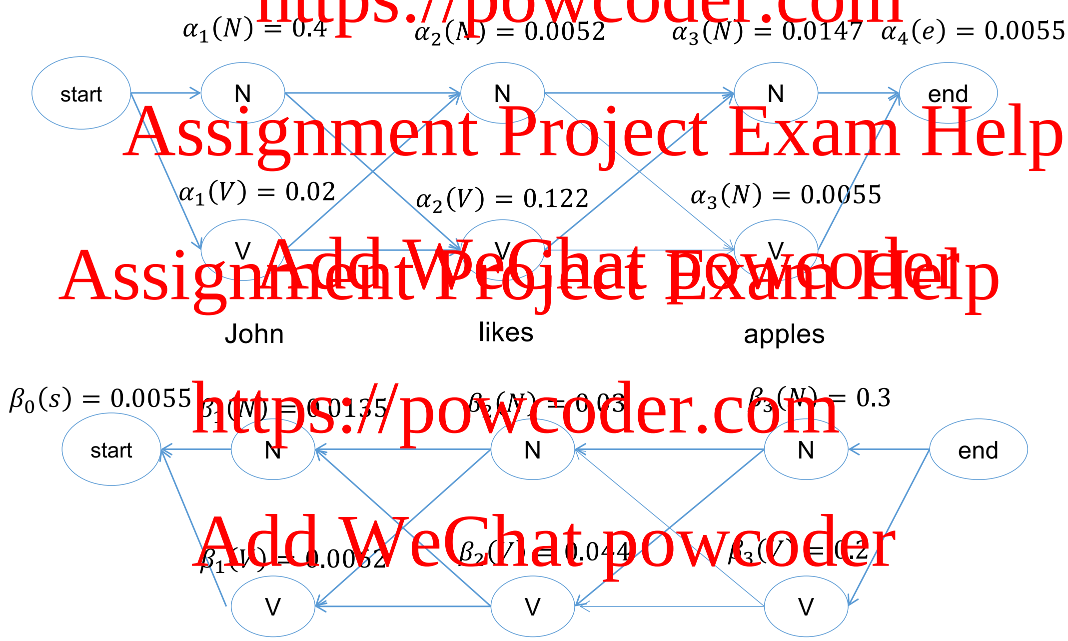
	Expected counts			probabilities		
	N	V	♦	N	V	♦
w_0=John	-	-	-	0.5	0.1	0
w_0=likes	-	-	-	0.1	0.5	0
w_0=apples	-	-	-	0.2	0.2	0
w_0=oranges	-	-	-	0.2	0.2	0
$t_{-1} = \Diamond$	-	-	-	0.8	0.2	0
t_{-1}=N	-	-	-	0.1	0.6	0.3
t_{-1}=V	-	-	-	0.6	0.2	0.2

Baum-Welch: Backward computation

	Expected counts			probabilities		
	N	V	♦	N	V	♦
w_0=John	-	-	-	0.5	0.1	0
w_0=likes	-	-	-	0.1	0.5	0
w_0=apples	-	-	-	0.2	0.2	0
w_0=oranges	-	-	-	0.2	0.2	0
$t_{-1} = \Diamond$	-	-	-	0.8	0.2	0
t_{-1}=N	-	-	-	0.1	0.6	0.3
t_{-1}=V	-	-	-	0.6	0.2	0.2

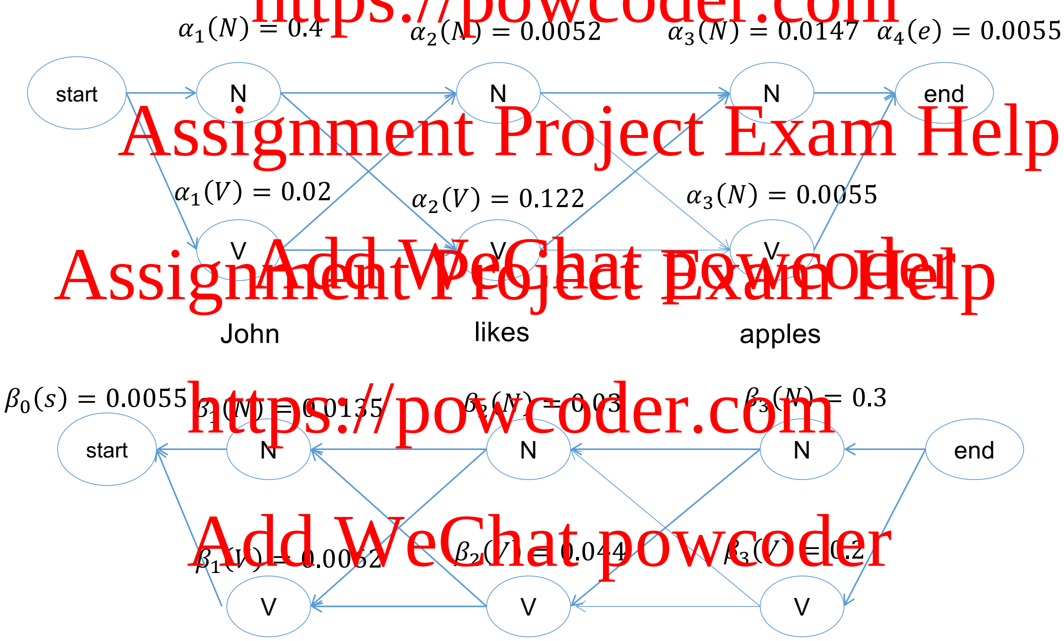


Collecting expected counts



$$
\begin{aligned}
\mathbb{E}[count(Y_2 = V, likes) | \text{John likes apples}] \\
= \alpha_2(V)\beta_2(V)/\alpha_4(end) = ?
\end{aligned}
$$

Collecting expected counts



$$\mathbb{E}[count(Y_2 = V, Y_1 = N) | \text{John likes apples}]$$
$$= \alpha_1(N) s_2(V, N) \beta_2(V) / \alpha_4(end) = ?$$

Sequence labeling summary

- Decoding: the Viterbi algorithm
- Parameter estimation

  - Supervised algorithms:
    - HMM: there is a closed form solution, assuming strong independence

    - Perceptron: The perceptron learning algorithm rewards features that make correct predictions and penalizes features that make incorrect predictions

    - CRF: Updates the feature weights proportionally, needs to compute expected feature counts, which in turns requires a posterior that can be computed efficiently with the forward backward algorithm.

    - LSTM-CRF: Learned feature representation and transition scores via RNNs.
  - Unsupervised algorithms:
    - The Baum-Welch algorithm, which combines expectation maximization and the forward-backward algorithm.