# CSC 246

**Spring 2020**

---

# 1 Lecture 1

## 1.1 Polynomial Curve Fitting

- Training data: $x = (x_1, \ldots, x_N)^T$.
- Target data: $t = (t_1, \ldots, t_N)^T$.
- Hypothesis space: $y(x, w) = \sum_{i=1}^{M} w_i x_i$.
- Error: $E(w) = \frac{1}{2} \sum_{i=1}^{N} (y(x_i, w) - t_i)^2$.
- Objective: $\widehat{w} = \arg\min_w E(w)$.
- Selecting $M$ (degree of the polynomial): Must balance trade-off between accuracy of the model and overfitting (encapsulating noise of the data into the model). The right balance allows our model to better generalize to new data. Note: More data yields less overfitting.
- Root mean squared error: $E_{RMS} = \sqrt{2E(w)/N}$. Division by $N$ allows meaningful comparison between datasets of different size. Square root causes the units to match the objective.
- Regularization: $\widetilde{E}(w) = E(w) + \frac{\lambda}{2} \|w\|^2$. Modify the error metric to prevent parameter values from "blowing up", thus limiting overfitting.
- Regression: Mapping inputs to real numbers.

## 1.2 Probability Review

- $\Pr(X, Y) = \Pr(Y|X) \Pr(X)$
- $\Pr(X|Y) = \frac{\Pr(Y|X) \Pr(X)}{\Pr(Y)}$

- $\mathcal{D} = \{t_1, \ldots, t_N\}$. Posterior probability:

$$\Pr(w|\mathcal{D}) = \frac{\Pr(\mathcal{D}|w)\Pr(w)}{\Pr(\mathcal{D})}.$$

  We call $\Pr(\mathcal{D}|w)$ the **likelihood function**. Computing $\arg\max_w \Pr(\mathcal{D}|w)$ is maximum likelihood estimation. Computing $\arg\max_w \Pr(w|\mathcal{D})$ is **maximum a posteriori**.

- $A \perp\!\!\!\perp B|C$ iff $\Pr(A|C)\Pr(B|C) = \Pr(A, B|C)$ iff $\Pr(A|B, C) = \Pr(A|C)$.

# 2 Lecture 2 - Classification

- Classification: Mapping inputs to categories.

- Linear Regression: Linear in the parameters. We assume the input features are transformed by fixed (not necessarily linear) basis functions $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_M)$. e.g. $\boldsymbol{\phi}_i(x) = x^i$, but we may also choose $\phi_i$ to be a set of Gaussian distributions or sigmoid distributions. So our model becomes

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\Phi}(\mathbf{x})$$

  where we take $\boldsymbol{\phi}_0 \equiv 1$. Note $\mathbf{x}$ is $D$-dimensional, $\mathbf{w}$ is $M$-dimensional and $\boldsymbol{\phi}$ is an $M$-dimensional vector of functions from $\mathbb{R}^D \to \mathbb{R}$.

- $p(t|\mathbf{X}, \mathbf{w}, \beta) = \prod \mathcal{N}(t_n|\mathbf{w}^T\boldsymbol{\phi}(x_n), \beta^{-1})$. Taking the log we get our error function (only part of dependent on $\mathbf{w}$):

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\boldsymbol{\phi}(x_n))^2.$$

  Taking the gradient with respect to $\mathbf{w}$,

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(t_n - \mathbf{w}^T\boldsymbol{\phi}(x_n))\boldsymbol{\phi}(x_n)^T$$

  Setting this to 0,

$$\sum_{n=1}^{N} t_n\boldsymbol{\phi}(x_n)^T = \mathbf{w}^T\sum_{n=1}^{N}\boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T$$

  Thus

$$\mathbf{w}_{\text{ML}} = (\boldsymbol{\Phi}(\mathbf{x})^T\boldsymbol{\Phi}(\mathbf{x}))^{-1}\boldsymbol{\Phi}(\boldsymbol{x})^T\mathbf{t}$$

  where row $i$ of $\boldsymbol{\Phi}$ is $\boldsymbol{\phi}(x_i)^T$.

# 3 Perceptrons

The goal of a perceptron is to learn a linear classification boundary, i.e. the normal vector to a hyperplane which correctly separates the training data into two classes.

- INPUT. $x_i \in \mathbb{R}^d$ and $t_i \in \{\pm 1\}$ (true classification).
- OBJECTIVE. $w \in \mathbb{R}^d$ maximizing $\sum_i I(y_i = t_i)$ where $y_i = \text{sign}(w^T x_i)$

**Algorithm.**

```
repeat
    for all i:
        if t_i ≠ y_i:
            w ← w + μ t_i x_i
until convergence or maxIter
```

**Convergence.** The perceptron algorithm converges for linearly separable data-sets. Suppose the data-set is linearly separable, i.e. there exists $u$ (the normal vector of the linear separator) such that $\|u\| = 1$ and $y_i u^T x_i \geq \delta$ for all $i$. Further, suppose $\|x_i\|^2 \leq R^2$ for all $i$.

Suppose $(x, y)$ is misclassified at step $k$. Then

$$w^{(k+1)} = w^{(k)} + yx \qquad \text{and} \qquad y(w^{(k)})^T x \leq 0. \tag{3.1}$$

Thus

$$\begin{aligned} u^T w^{(k+1)} &= u^T w^{(k)} + y u^T x \\ &\geq u^T w^{(k)} + \delta \\ &\geq k\delta. \end{aligned}$$

Hence $\left\| u^T w^{(k+1)} \right\|^2 \geq k^2\delta^2$. By Cauchy-Schwarz, $k^2\delta^2 \leq \|u\|^2 \left\| w^{(k+1)} \right\|^2 = \left\| w^{(k+1)} \right\|^2$. On the other hand,

$$\begin{aligned}\left\|w^{(k+1)}\right\|^2 &= \left\|w^{(k)} + yx\right\|^2 \\ &= \left\|w^{(k)}\right\|^2 + \|y\|^2 \|x\|^2 + 2y(w^{(k)})^T x \\ &\leq \left\|w^{(k)}\right\|^2 + \|y\|^2 \|x\|^2 \\ &\leq \left\|w^{(k)}\right\|^2 + R^2 \\ &\leq kR^2.\end{aligned}$$

Whence, $k \leq \frac{R^2}{\delta^2}$.

# 4 Binary Logistic Regression

**Sigmoid Function.**

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}. \tag{4.1}$$

Note

$$\frac{\mathrm{d}\sigma(x)}{\mathrm{d}x} = \sigma(x)(1-\sigma(x)) \tag{4.2}$$

Assume we have some feature vector of basis functions $\boldsymbol{\phi} = (\psi_1, \ldots, \psi_d)$. Let $\boldsymbol{\phi}_n = \boldsymbol{\phi}(x_n)$ for $n \in [N]$ and true classifications $\boldsymbol{t} = (t_1, \ldots, t_N)$.

We interpret $\Pr(\mathcal{C}_1|\boldsymbol{\phi_n}) = y_n := \sigma(w^T\boldsymbol{\phi_n})$. Thus $\Pr(\mathcal{C}_2|\boldsymbol{\phi_n}) = 1 - y_n$. We can write the likelihood as

$$\Pr(\boldsymbol{t}|w) = \prod_{n=1}^{N} y_n^{t_n}(1-y_n)^{1-t_n}.$$

Thus

$$E(w) = -\ln \Pr(\boldsymbol{t}|w) = -\sum_{n=1}^{N} t_n \ln y_n + (1-t_n)\ln(1-y_n).$$

The above expression is called cross entropy. Calculating the gradient with respect to $w$:

$$\nabla E(w) = \sum_{n=1}^{N}(y_n - t_n)\boldsymbol{\phi_n}^T.$$

There is no analytic solution for setting this to zero, so we apply gradient descent.

**Gradient Descent Algorithm.**

```
1  w_0 ← random initial weights
2  while t < maxIters and ||E(w_t)||^2 > ε:
3      w_{t+1} ← w_t − α∇E(w_t) // α is the learning rate
```

Can either do batch gradient descent as above iterating over all data points, or stochastic gradient descent iterating over a random selection of data points (better at escaping local minima).

### 4.1 $K$-Class Logistic Regression

We have $\Pr(\mathcal{C}_k|\boldsymbol{\phi}) = y_n(\boldsymbol{\phi}) := \frac{\exp(a_k)}{\sum_j \exp(a_j)}$ (here $y_n$ is the softmax function) where $a_k = w_k^T \boldsymbol{\phi}$. Define $y_{nk} = y_k(\boldsymbol{\phi_n})$. The likelihood function is

$$\Pr(\boldsymbol{t}|\boldsymbol{w_1}, \ldots, \boldsymbol{w_k}) = \prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}}.$$

Thus

$$E(w) = -\sum_{n,k} t_{nk} \ln y_{nk}.$$

Note $\sum_k t_{nk} = 1$ (any data point has exactly one class) and $\frac{\mathrm{d}y_k}{\mathrm{d}a_j} = y_k(I_{kj} - y_j)$.

$$\nabla_{\boldsymbol{w_j}} E(\boldsymbol{w_1}, \ldots, \boldsymbol{w_K}) = \sum_{n=1}^{N} (y_{nj} - t_{nj}) \boldsymbol{\phi_n}.$$

## 5 Neural Networks

Activation Functions:

- $\tanh(x)$
- $\sigma(x)$
- $\mathrm{ReLU}(x)$

Suppose $k_n$ is the number of hidden units in the $n$-th hidden layer, $n = 1, \ldots, N$. Let $k_0 = d_{in}$ and $k_{N+1} = d_{out}$.

- Let $W_{ji}^{(n)}$ denote the weight from node $i$ in layer $n-1$ to node $j$ in layer $n$ for $n = 1, \ldots, N+1$, $i \in [k_{n-1}]$, $j \in [k_n]$. Hence $W^{(n)}$ is a $k_n \times k_{n-1}$ matrix.
- Let $\widehat{W^{(n)}}$ be $W^{(n)}$ with an extra column for biases. So $\widehat{W^{(n)}}$ is $k_n \times (k_{n-1}+1)$.
- Let $\hat{o} = (x_1, \ldots, x_{d_{in}}, 1)$ denote the input extended with a 1 for biases.
- Then $o^{(n)} = h(\widehat{W^{(n)}} \cdot \widehat{o^{(n-1)}})$, where the activation function $h$ is applied componentwise (again $\widehat{o^{(n-1)}}$ is $o^{(n-1)}$ with a 1 adjoined). In summation notation,

$$o_i^{(n)} = h\left(\sum_{j=1}^{k_{n-1}+1} \widehat{W^{(n)}}_{ij} \widehat{o^{(n-1)}}_j\right)$$

  We'll use $a^{(n)}$ to denote the product $\widehat{W^{(n)}} \cdot \widehat{o^{(n-1)}}$. So

$$a_k^{(n)} = \sum_j \widehat{W^{(n)}}_{kj} \widehat{o^{(n-1)}}_j = \sum_j \widehat{W^{(n)}}_{kj} h(\widehat{a_j^{(n-1)}})$$

## 5.1 Backpropagation

By the chain rule,

$$\frac{\partial E}{\partial W_{kj}^{(n)}} = \frac{\partial E}{\partial a_k^{(n)}} \cdot \frac{\partial a_k^{(n)}}{\partial W_{kj}^{(n)}} = \delta_k^{(n)} \cdot \frac{\partial a_k^{(n)}}{\partial W_{kj}^{(n)}}.$$

Note

$$\frac{\partial a_k^{(n)}}{\partial W_{kj}^{(n)}} = h(a_j^{(n-1)}).$$

Now using the total derivative, we have

$$\begin{aligned}\delta_k^{(n)} := \frac{\partial E}{\partial a_k^{(n)}} &= \sum_i \frac{\partial E}{\partial a_i^{(n+1)}} \cdot \frac{\partial a_i^{(n+1)}}{\partial a_k^{(n)}} \\ &= \sum_i \delta_i^{(n+1)} \cdot \frac{\partial a_i^{(n+1)}}{\partial a_k^{(n)}}\end{aligned}$$

where

$$\frac{\partial a_i^{(n+1)}}{\partial a_k^{(n)}} = W_{ik}^{(n+1)} h'(a_k^{(n)}).$$

Hence

$$\delta_k^{(n)} = h'(a_k^{(n)}) \sum_i \delta_i^{(n+1)} W_{ik}^{(n+1)}.$$

We can express this in matrix notation:

$$D^{(n)} = \text{Diag}(h'(a_1^{(n)}), \ldots, h'(a_{k_n}^{(n)})), \quad \dim(D^{(n)}) = [k_n \times k_n]$$

$$(\delta^{(n)})^T = (\delta^{(n+1)})^T W^{(n+1)} D^{(n)}, \quad \dim(\delta^{(n)}) = [k_n \times 1]$$

$$\widehat{W^{(n)}} \leftarrow \widehat{W^{(n)}} - \alpha \delta^{(n)} \widehat{o^{(n-1)}}^T, \quad \dim(o^{(n-1)}) = [k_{n-1} \times 1].$$

Bishop assumes output activation is the identity. Assuming the error function is sum of squares, $E(w) = \frac{1}{2} \sum_i (a_i^{(N+1)} - t_i)^2$, we have

$$\delta^{(N+1)} = \begin{pmatrix} a_1^{(N+1)} - t_1 \\ \vdots \\ a_{k_{N+1}}^{(N+1)} - t_{k_{N+1}} \end{pmatrix}$$

# 6 Deep Learning

Convolutional Neural Networks.

- Significantly fewer parameters than a fully connected network.
- Pooling layers lose information and prevent overfitting.
- Convolution is linear and shift-invariant.
- "Shifting dot product"

Controlling overfitting.

- Data Augmentation. Apply label-preserving transformations, e.g. translations and reflections of the image.
- Dropout. Randomly set activation to 0 during training with probability 1/2. At test time multiply activations by 1/2.

# 7 Support Vector Machines

Our goal is to find a decision boundary that maximizes the distance between the boundary and the nearest data point. The data points at the boundary are called the **support vectors**. Our model is given by

$$y(x) = w^T \boldsymbol{\phi}(x) + b.$$

We have $w^T(x_A - x_B) = 2$, thus $\|w\| \, \|x_A - x_B\| \cos\theta = 2$. On the other hand, $\|x_A - x_B\| \cos\theta = M$, so $M = \frac{2}{\|w\|}$. Thus, maximizing $M$ is equivalent to maximizing $\frac{2}{\|w\|}$.

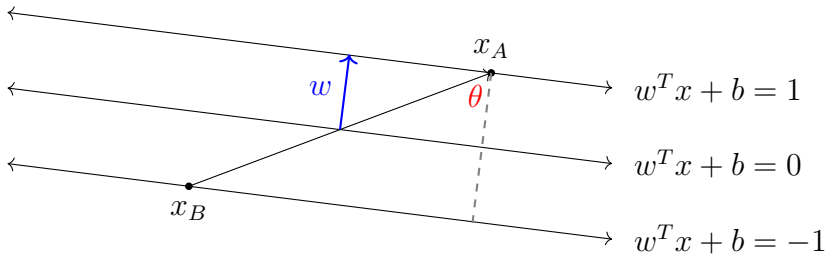



Figure 1: Relating max separator to min weight.

Our objective is to $\min_w \frac{1}{2} \|w\|^2$ subject to $t_n y_n = t_n(w^T \boldsymbol{\phi}(x_n) + b) \geq 1$ for all $n$.

If our data is non-linearly separable, we need to punish misclassifications. Introduce slack variables $\xi_n \geq 0$ for each data point.

$$\xi_n = \begin{cases} 0 & \text{if } t_n y_n \geq 1 \\ |t_n - y_n| & \text{otherwise} \end{cases} = \max\left\{0, 1 - t_n(w^T \boldsymbol{\phi}(x_n) + b)\right\}$$

Note $t_n y_n \leq 1$ (i.e. a misclassification) is equivalent to $1 - t_n y_n \geq 0$. Since $t_n \in \{\pm 1\}$, we thus have $|t_n - y_n| = |1 - t_n y_n| = 1 - t_n y_n$.

We obtain our primal optimization problem:

$$\arg\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{N} \xi_n, \text{ subject to}$$
$$t_n(w^T \boldsymbol{\phi}(x_n) + b) \geq 1$$
$$\xi_n \geq 0$$

## 7.1 Lagrange Multipliers

Suppose we want to maximize a function $f(\boldsymbol{x})$ subject to the constraint $g(\boldsymbol{x}) = 0$. We define the Lagrangian

$$L(\boldsymbol{x}, \lambda) := f(\boldsymbol{x}) + \lambda g(\boldsymbol{x})$$

where we require $\nabla_{\boldsymbol{x}} L = 0$ and $\frac{\partial L}{\partial \lambda} = 0$ (for minimization problems we simply use $-\lambda$ instead).

Consider the level curve given by $g(\boldsymbol{x}) = 0$. Along the level curve, $\nabla_{\boldsymbol{x}} g$ will be orthogonal to the level curve as otherwise the value of $g$ wouldn't be constant along the curve. Similarly, if $f$ obtains it's maximum or minimum value of the level curve at $\boldsymbol{x_0}$, then $\nabla_{\boldsymbol{x}} f$ is orthogonal to the level curve at $\boldsymbol{x_0}$, as otherwise, we could further optimize $f$ by stepping along the curve. Thus, we obtain $\nabla f + \lambda \nabla g = 0$ for some $\lambda \neq 0$.

Suppose we allow $g(\boldsymbol{x}) \geq 0$. Let $S = \{\boldsymbol{x} : g(\boldsymbol{x}) \geq 0\}$, then

- Case 1. $f$ is maximized on the interior of $S$. At such a point $\nabla f = 0$, $\lambda = 0$ and $g(\boldsymbol{x}) > 0$.
- Case 2. The maximum of $f$ occurs on the boundary, i.e. $g(\boldsymbol{x}) = 0$ and $\nabla g$ points towards $S$.

  Note: $f$ is a maximum on the boundary only when $\nabla f$ is oriented away from $S$ (else we'd be in case 1), so necessarily $\nabla f = -\lambda \nabla g$ for some $\lambda > 0$. $f$ is a minimum only when $\nabla f$ is oriented towards $S$, so $\nabla f = \lambda \nabla g$, $\lambda > 0$.

In any case $g(\boldsymbol{x})\lambda = 0$. We obtain the Karush-Kuhn-Tucker (KKT) conditions. Maximizing $f$ subject to $g \geq 0$ is equivalent to maximizing $L(x, \lambda)$ subject to

$$\begin{gathered} g(x) \geq 0 \\ \lambda \geq 0 \\ \lambda g(x) = 0. \end{gathered}$$

The general procedure is as follows:

- To maximize $f$ subject to $g_j(x) = 0$ and $h_k \geq 0$ we introduce Lagrange multipliers $\{\lambda_j\}, \{\mu_k\}$.
- The Lagrangian is $f(x) + \sum_j \lambda_j g_j(x) + \sum_k \mu_k h_k(x)$, which we maximize subject to $\mu_k \geq 0$ and $\mu_k h_k(x) = 0$.

## 7.2 Apply Lagrange to SVM

The Lagrangian corresponding to our primal is

$$L(w, b, \boldsymbol{\alpha}) = \frac{1}{2}\|w\|^2 + C\sum_n \xi_n - \sum_n \alpha_n(t_n(w^T\boldsymbol{\phi}(x_n) + b) + \xi_n - 1) - \sum_n \mu_n\xi_n. \quad (7.1)$$

subject to

$$\alpha_n, \xi_n, \mu_n \geq 0 \quad (7.2)$$
$$t_n(w^T\boldsymbol{\phi}(x_n) + b) - \xi_n - 1 \geq 0 \quad (7.3)$$
$$\alpha_n(t_n(w^T\boldsymbol{\phi}(x_n) + b) - \xi_n - 1) = 0 \quad (7.4)$$
$$\mu_n\xi_n = 0 \quad (7.5)$$

By taking the gradient of $L$ wrt $w$ we obtain

$$w = \sum_n \alpha_n t_n \boldsymbol{\phi}(x_n).$$

Taking it wrt $b$, we obtain

$$\sum_n \alpha_n t_n = 0.$$

Taking it wrt $\xi_n$, we obtain

$$\alpha_n + \mu_n = C.$$

Substituting these conditions into the Lagrangian, we obtain the dual

$$\max_{\boldsymbol{\alpha}} \sum_n \alpha_n - \frac{1}{2}\sum_{n,m} \alpha_n\alpha_m t_n t_m \boldsymbol{\phi}(x_n)^T\boldsymbol{\phi}(x_m)$$

subject to $0 \leq \alpha_n \leq C$ (this condition follows since $0 \leq \alpha_n = C - \mu_n \leq C$). This dual is easy to solve. We typically vary to $\alpha$'s at a time and numerically optimize the dual, then use $\boldsymbol{\alpha}^*$ to get $w^*$ from (7.2).

In matrix notation,

$$\operatorname*{argmax}_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^T(\mathbf{1}_n + A\boldsymbol{\alpha}), 0 \leq \boldsymbol{\alpha} \leq C$$

where $A_{ij} = t_i t_j \boldsymbol{\phi}(x_i)\boldsymbol{\phi}(x_j)^T$ and $\mathbf{1}_n$ is a $n \times 1$ vector of 1's.

**Kernel Trick.** Note, we don't explicitly need $\boldsymbol{\phi}$, we just need the dot product $\boldsymbol{\phi}(x_n)^T\boldsymbol{\phi}(x_m)$. We really only need to know $K(x_n, x_m) = \boldsymbol{\phi}(x_n)^T\boldsymbol{\phi}(x_m)$.

- $K(x_n, x_m) = (x_n^T x_m + 1)^k$.
- $K(x_n, x_m) = \exp(-\|x_n - x_m\|^2/2\sigma^2)$.

# 8 Hidden Markov Model

- $\Omega$ set of states (value of hidden variables), $y_i \in \Omega$
- $\Sigma$ set of possible emissions (value of observable variable), $x_j \in \Sigma$
- $P \in [0,1]^{|\Omega|\times|\Omega|}$. $P(y_i, y_j)$ is the probability of transition from state $y_i$ to state $y_j$ [First order homogeneous Markov assumption]
- $Q \in [0,1]^{|\Omega|\times|\Sigma|}$. $Q(y_i, x_j)$ is the probability of emitting $x_j$ in $y_i$.
- $\Pi \in [0,1]^{|\Omega|}$, $\Pi(y)$ is the probability of starting in state $y$.

$$\Pr(x_1, \ldots, x_n, y_1, \ldots, y_n) = \Pi(y_1) \prod_{i=1}^{n-1} P(y_i, y_{i+1}) \prod_{i=1}^{n} Q(y_i, x_i).$$

Dynamic Programming.

$$\alpha^t(a) = \Pr(x_1, \ldots, x_t, y_t = a) \tag{8.1}$$

$$\alpha^{t+1}(a) = \sum_{c\in\Omega} \alpha^t(c) P(c,a) Q(a, x_{t+1}), \text{ where } \alpha^1(a) = \Pi(a) Q(a, x_1) \tag{8.2}$$

$$\tag{8.3}$$

Here $\alpha$ (forward) is the probability of state $a$ in time $t$ and observation sequence $x_i$. $\beta$ is the probability of future observation sequence given in state $a$ at time $t$ (backward).

$$\beta^t(a) = \Pr(x_{t+1}, \ldots, x_n | y_t = a) \tag{8.4}$$

$$\beta^{t-1} = \sum_{c\in\Omega} Q(c, x_t) \beta^t(c) P(a,c) \text{ where } \beta^n(a) = 1 \tag{8.5}$$

**Evaluate likelihood of observation sequence.**

$$p(x_1, \ldots, x_n) = \sum_{a\in\Omega} \alpha^n(a) = \sum_{a\in\Omega} \Pi(a) Q(a, x_1) \beta^1(a) \tag{8.6}$$

**Revise, i.e. determine** $\Pr(y_t = a | x_1, \ldots, x_n)$**.** We'll make use of the fact that

$X_{t+1}^N \perp\!\!\!\perp X_1^t | y_t$.

$$\begin{aligned}\Pr(y_t = a|X_1^N) &= \frac{\Pr(X_1^N|y_t = a)\Pr(y_t = a)}{\Pr(X_1^N)} \\ &= \frac{\Pr(X_{t+1}^N|y_t = a)\Pr(X_1^t|y_t = a)\Pr(y_t = a)}{\Pr(X_1^N)} \\ &= \frac{\Pr(X_{t+1}^N|y_t = a)\Pr(X_1^T, y_t = a)}{\Pr(X_1^N)} = \frac{\alpha^t(a)\beta^t(a)}{\Pr(X_1^N)} \\ &= \frac{\alpha^t(a)\beta^t(a)}{\sum_{c\in\Omega}\alpha^t(c)\beta^t(c)}\end{aligned} \tag{8.7}$$

**Given $x_1, \ldots, x_n, P, Q, \Pi$ determine $y_1, \ldots, y_n$ to maximize expected number of correct states.** Take $\widehat{y_t} = \text{argmax}_{c\in\Omega}\Pr(y_t = c|X_1^N)$.

**Given $x_1, \ldots, x_n, P, Q, \Pi$ determine $y_1, \ldots, y_n$ to maximize $\Pr(x_1, \ldots, x_1, y_1, \ldots, y_n)$.** Viterbi's Algorithm.

$$\max_{y_1,\ldots,y_{t-1}} \Pr(y_1, \ldots, y_t = a, x_1 \ldots, x_t)$$

$$\delta^{t+1}(a) = Q(a, x_{t+1})\left(\max_{c\in\Omega}\delta^t(c)P(c, a)\right), \delta^1(a) = \Pi(a)Q(a, x_1) \tag{8.8}$$

$$\phi^t(a) = \operatorname*{argmax}_{c\in\Omega}\delta^{t-1}(c)P(c, a) \tag{8.9}$$

$$P^* = \max_{a\in\Omega}\delta^T(a)$$

$$q_T^* = \operatorname*{argmax}_{a\in\Omega}\delta^T(a)$$

$q_t^* = \phi^t(q_{t+1}^*)$

# 9 Graphical Models

$$\Pr(X_1, \ldots, X_n) = \prod_{i=1}^N \Pr(X_i|\text{parents}(X_i)). \tag{9.1}$$

The independence assumptions encoded in the network allow us to drastically reduce the number of parameters. If we have $N$ variables, each taking $\leq I$ values, and each variable has $\leq M$ parents, we reduce from $I^N$ parameters to $N \cdot I^{M+1}$ parameters.

We can perform inference as follows:

$$\Pr(X_i|\vec{X}_e) = \frac{1}{Z}\sum_{Y} \Pr(X_i, \vec{Y}, \vec{X}_e)$$

where we're summing over all possible values of non-evidence, non-query variables $\vec{Y}$ and

$$Z = \sum_{x_i} \Pr(X_i = x_i, \vec{X}_e)$$

is the normalization constant.

## 9.1 Bayes' Ball

**Case 1.** Arrows meet head-to-tail or tail-to-head.

$$A \longrightarrow \mathbf{B} \longrightarrow C$$

By the independence conditions in the network (and Bayes' rule), $A, C$ are conditionally independent given $B$:

$$P(A, C|B) = \frac{\Pr(A, B, C)}{\Pr(B)} = \frac{\Pr(A)\Pr(B|A)\Pr(C|B)}{\Pr(B)} = \Pr(A|B)\Pr(C|B).$$

**Case 2.** Arrows meet tail-to-tail.

$$A \longleftarrow \mathbf{B} \longrightarrow C$$

Then $A, C$ are conditionally independent given $B$:

$$\Pr(A, C|B) = \frac{\Pr(A, B, C)}{\Pr(B)} = \frac{\Pr(B)\Pr(A|B)\Pr(C|B)}{\Pr(B)} = \Pr(A|B)\Pr(C|B).$$

**Case 3**. Arrows meet head-to-head (we don't condition on the middle).

$$A \longrightarrow B \longleftarrow C$$

Then $A$ and $C$ are independent.

$$\begin{aligned}\Pr(A, C) &= \sum_b \Pr(A, b, C) = \sum_b \Pr(A)\Pr(C)\Pr(b|A, C) \\ &= \Pr(A)\Pr(C)\sum_b \Pr(b|A, C) = \Pr(A)\Pr(C).\end{aligned}$$

## 9.2 D-separation

**Definition.** Let $A, B, C$ be sets of nodes. An $A, B$-path in the network is an undirected path from a node in $A$ to a node in $B$. We say an $A, B$-path is **blocked** by $C$ if it includes a node such that

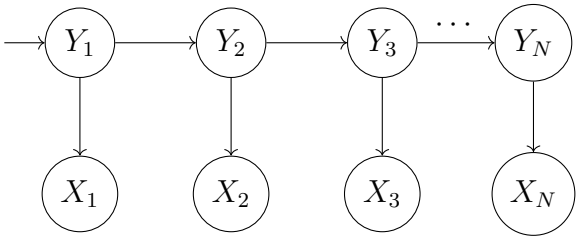
a) arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in $C$, or

b) arrows meet head-to-head, and neither the node nor any of its descendants are in $C$.

If all $A, B$-paths are blocked by $C$, we say $A$ is **d-separated** from $B$ by $C$, in which case $A$ is conditionally independent of $B$ given $C$.

## 9.3 Independence Conditions of an HMM

The HMM conditions correspond to the following network:




By applying case 1 from above, we have $Y_N$ is conditionally independent of $\{Y_1, \ldots, Y_{n-1}\}$ given $Y_n$. Likewise,

$$(Y_N \perp\!\!\!\perp Y_1) | (Y_n, X_1, \ldots, X_N)$$

# 10 Undirected Graphical Models

We introduce factors $f_m$, e.g. $f_1(x_1, x_3, x_4) = \exp(w_1 x_1 + w_3 x_3 + w_4 x_4)$.

$$\Pr(X_1, \ldots, X_N) = \frac{1}{Z} \prod_{m=1}^{M} f_m(\vec{x_m}) \tag{10.1}$$

where $\vec{x_m}$ denotes the set of arguments to factor $f_m$ and $Z$ is the normalization constant.

## 10.1 Undirected to Directed

**Moralization**: For each node in the graph, add an undirected link between all pairs of parents of that node. Replace all directed edges with undirected edges. Identify the clique potentials in the resulting MRF with the conditional probabilities in the directed model.

If every conditional independence relation of a distribution is reflected in a given graph, the graph is a **D map** of the distribution. Ex. Edgeless graph is a D map for any distribution.

If every conditional independence relation implied by a graph is present in a specific distribution, the graph is an **I map** for the distribution. Ex. $K_n$ is an I map for any distribution.

A graph which is an I map and D map for a distribution is a **perfect map** for that distribution.

Let $D$ be the set of distributions for which there is a perfect, directed map and let $U$ be the set of distributions for which there is a perfect, undirected map. There is some $x \in U$ such that $x \notin D$ and some $y \in D$ such that $y \notin U$.

See written notes for inference in a chain.

## 10.2 Factor Graph

## 10.3 Addition

# 11 EM

**Learning.** Given sequences of observations, learn $P, Q, \Pi$ to maximize the likelihood of observed data.

**Matrix Convolution Example**

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \star \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 7 & 10 & 9 \\ 4 & 16 & 19 & 15 \\ 7 & 15 & 17 & 9 \end{bmatrix}$$

# Contents