CS489: Machine Learning

Michael Noukhovitch

Winter 2017,

Notes written from Pascal Poupart's lectures.

Contents

1 Introduction

machine learning giving computers ability to learn without being explicitly programmed

A machine learns from experience E wrt to some class of tasks T and performance measure P if its performance in task T, as measured by P, improves with E

Three types:

- supervised
- unsupervised
- reinforcement

Long-term goals:

- meta-programming
- lifelong machine learning
- transfer learning

1.1 Supervised Learning

Definition. given a training set of examples $(x, f(x))$, return a hypothesis h that approximates h

Two types:

classification where output space consists of *categorical* values

regression where output space consists of *numerical* values

1.1.1 Hypothesis Space

hypothesis space set of all hypotheses H that the learner may consider

consistent if hypothesis h agrees with f on all examples

realizable if the hypothesis space constains the consistent function

our objective can be restated as a search problem to find the hypothesis h in hypothesis space H that minimizes some objective

2 Nearest Neighbour

2.1 Basic NN

nearest neighbours label any example with the label of its nearest neighbours

classification: $h(x) = y_{x*}$
where $y_{x*} = \text{argmin}_{x'} d(x, x')$ is the label associated with the nearest neighbour

2.2 KNN

k-nearest neighbours assign the most frequent label among k nearest neighbours

let $knn(x)$ be the k nearest neighbours
then $y_x = \text{mode}(y_{x'}|x' \in knn(x))$

overfitting a hypothesis h with training accuracy higher than its own testing accuracy $\max(0, \text{trainAccuracy}(h) - \text{testAccuracy}(h))$

- classifier too expressive
- noisy data
- lack of data

underfitting a hypothesis h with training accuracy lower than testing accuracy of some other hypothesis h', $\max(0, \max_{h'} \text{trainAccuracy}(h) - \text{testAccuracy}(h'))$

- classifier not expressive enough

***k*-fold cross validation** split data in k equal subsets, run k experiments testing on one subset, and training on all the others. Report average accuracy

weighted knn weight each neighbour by distance

knn regression y_x is a real value, $y_x \leftarrow average(y_{x'}|x' \in knn(x))$

properties:

+ lazy learning (learn on test)
+ concpetually simple
+ flexible decision boundaries
- good distance measures are hard to find
- noise can strongly influence
- can't handle more than a few dozen attributes/features
- requires a lot of computation and memory

3 Linear Regression

3.1 Least Squares

find linear hypothesis h: $t = w^T\bar{x}$, find w to minimize euclidean L2 loss

$$w* = argmin_w \frac{1}{2}\sum_{n=1}^{N}(t_n - w^T\bar{x}_n)^2$$

where $\bar{x} = \begin{pmatrix} 1 \\ x \end{pmatrix}$ and we can solve with $w = A^{-1}b$ or $Aw = b$ which can be solved as a linear system

3.1.1 Regularization

Least squares can be unstable, overfit, so change optimization

$$w* = argmin_w \frac{1}{2}\sum_{n=1}^{N}(t_n - w^T\bar{x}_n)^2 + \frac{\lambda}{2}\|w\|_2^2$$

or $(\lambda I + A)w = b$

3.2 Maximum Likelihood

derive the same thing but from a different perspective: assume $y = w^T\bar{x} +$ gaussian noise so

$$\begin{aligned} Pr(y|\bar{X}, w, \sigma) &= N(y|w^T\bar{X}, \sigma^2) \\ w* &= argmax_w Pr(y|\bar{X}, w, \sigma) \\ &... \\ &= argmin_w \sum_n (y_n - w^T\bar{x}_n)^2 \end{aligned}$$

which is the same as least squares

3.3 Maximum A Posteriori

find $w*$ with highest posterior probability, knowing that prior $P(w) = N(0, \Sigma)$

$$Pr(w|X, y) \propto Pr(w)Pr(y|X, w)$$

therefore for optimization:

$$\begin{aligned} w* &= argmax_w Pr(w|\bar{X}, y) \\ &... \\ &= argmin_w \sum_n (y_n - w^T\bar{x}_n)^2 + w^T\Sigma^{-1}w \end{aligned}$$

let $\Sigma^{-1} = \lambda I$

$$= argmin_w \sum_n (y_n - w^T\bar{x}_n)^2 + \lambda\|w\|_2^2$$

and we arrive at least squares with regularization

3.4 Expected Squared Loss

$$\begin{aligned} E[loss] &= \int_{x,y} Pr(x, y)(y - w^T\bar{x})^2 dxdy \\ &= \int_{x,y} Pr(x, y)(y - f(x))^2 + \int_x Pr(x)(f(x) - w^T\bar{x})^2 dx \\ &= \text{noise (constant)} + \text{error (relative to } w) \end{aligned}$$

lets consider the expected error wrt our dataset S

$$\begin{aligned}
E[error] &= E_S[(f(x) - w_S^T)^2] \\
&= (f(x) - E_S[w_S^T \bar{x}])^2 + E_S[(E_S[w_S^T \bar{x}] - w_S^T \bar{x})^2] \\
&= \text{bias}^2 + \text{variance}
\end{aligned}$$

therefore putting it together

$$E[loss] = \text{bias}^2 + \text{variance} + \text{noise}$$

3.5 Bayesian Linear Regression

instead of using $w*$, compute weighted avg prediction using $Pr(w|\bar{X}, y)$

$$Pr(w|\bar{X}, y) = N(\bar{w}, A^{-1})$$

where $w = \sigma^{-2} A^{-1} \bar{X}^T y$
$A = \sigma^{-2} \bar{X}^T \bar{X} + \Sigma^{-1}$

3.6 Bayesian Prediction

let x_* be the input for which we predict y_*

$$\begin{aligned}
Pr(y_*|\bar{x}_*, \bar{X}, y) &= \int_w Pr(y_*|\bar{x}_*, w) Pr(w|\bar{X}, y) dw \\
&\ldots \\
&= N(\bar{x}_*^T A^{-1} \bar{X}^T y, \bar{x}_*^T A^{-1} \bar{x}_*)
\end{aligned}$$

4 Statistical Learning

4.1 Introduction

probability distribution a specific probability for each event in our sample space

joint distribution spec of probabilities for all combinations of events $Pr(A \wedge B)$

conditional probabilties $Pr(A|B) = Pr(A \wedge B)/Pr(B)$

4.2 Bayes Rules

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

posterior P(B|A)

likelihood P(A|B)

prior P(B)

normalizing P(A)

evidence A

4.3 Bayesian Learning

computing the posterior of hypothesis given evidence using Bayes' theorem:

$$Pr(H|e) = kPr(e|H)Pr(H)$$

where k is a normalization constant such that the sum of all $Pr(H|e) = 1$
properties:

- \+ optimal (given prior)
- \+ no overfitting (all hypotheses considered)
- \- intractable if hypothesis space is large

prediction:

$$Pr(X|e) = \sum_i Pr(X|h_i)P(h_i|e)$$

4.4 Approximate Bayesian Learning

Maximum A Posteriori make prediction based on most probable hypothesis (vs basing on all hypotheses weighted by probability)

$$\begin{aligned} h_{map} &= argmax_{h_i} Pr(h_i|e) \\ &= argmax_{h_i} Pr(e|h_i)Pr(h) \\ Pr(X|e) &= Pr(X|h_{map}) \end{aligned}$$

- \+ controlled overfitting
- \+ converges as data increases
- \- less accurate than Bayesian prediction
- \- maybe be intractable!

Maximum Likelihood simplify MAP by assuming uniform prior $Pr(h_i) = Pr(h_j) \forall i, j$

$$\begin{aligned} h_{ml} &= argmax_{h_i} Pr(e|h_i) \\ Pr(X|e) &= Pr(X|h_{ml}) \end{aligned}$$

- \+ still converges
- \- least accurate because ignore prior info
- \- overfits

also, can be easier than MAP: $h_{ml} = argmax_h \sum_n \log Pr(e_n|h)$

5 Mixture of Gaussians

5.1 Introduction

Assume:

- each prior is frequency: $Pr(C = c_k) = \pi_k$
- $Pr(x|C)$ is gaussian
- covariance matrix Σ is used for each class

We can use maximum likelihood to estimate the parameters:

$$Pr(x|c_k) \propto e^{-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1}(x-\mu_k)}$$

Where:

$$\begin{aligned}
\pi &= \frac{\sum_n y_n}{N} && \text{average } y \\
\mu_k &= \frac{\sum_{n \in c_k} x_n}{N_k} && \text{mean of class } k \\
\Sigma &= \frac{N_1}{N} S_1 + \frac{N_2}{N} S_2 \ldots && \text{covariance} \\
S_k &= \frac{1}{N_k} \sum_{n \in c_k} (x_n - \mu_k)(x_n - \mu_k)^T && \text{weighted variance}
\end{aligned}$$

5.2 Two Class

Then if there are two classes c_k and c_j,

$$\begin{aligned}
Pr(c_k|x) &= \frac{1}{1 + e^{-(w^T x + w_0)}} \\
&= \sigma(w^T x + w_0)
\end{aligned}$$

$$\begin{aligned}
\text{where } w &= \Sigma^{-1}(\mu_k - \mu_j) \\
w_0 &= -\frac{1}{2}\mu_k^T \Sigma^{-1} \mu_k + \frac{1}{2}\mu_j^T \Sigma^{-1} \mu_j + \ln \frac{\pi_k}{\pi_j}
\end{aligned}$$

choose the best class as the one with probability > 0.5, so class boundary is at

$$\begin{aligned}
\sigma(w_k^T x + w_0) &= 0.5 \\
w_k^T \bar{x} &= 0 \text{ is a linear separator}
\end{aligned}$$

5.3 Multi-class

Normalize using softmax:

$$
\begin{aligned}
Pr(c_k|x) &= \frac{Pr(c_k)Pr(x|c_k)}{\sum_j Pr(c_j)Pr(x|c_j)} \\
&= \frac{e^{w_k^T \bar{x}}}{\sum_j e^{w_j^T \bar{x}}}
\end{aligned}
$$

6 Logistic Regression

6.1 Introduction

MoG is restrictive, assumes everything is a gaussian. Generalize to exponential family:

$$Pr(x|\Theta_k) = \exp(\Theta_k^T T(x) - A(\Theta_k) + B(x))$$

where Θ_k : parameters of class k

$T(x), A(\Theta_k), B(x)$: arbitrary functions

and the posterior $Pr(c_k|x) = \sigma(w^T x + w_0)$ which we will learn directly by maximum likelihood, in general it is

- **logistic sigmoid** for binary
- **softmax** for multiclass

6.2 Logisitc Regression Classification

For some dataset (X, y) and for two classes $y \in 0, 1$:

$$w^* = argmax_w \prod_n \sigma(w^T \bar{x}_n)^{y_n} (1 - \sigma(w^T \bar{x}_n))^{1-y_n}$$

so our objective is

$$L(w) = -\sum_n y_n \ln \sigma(w^T \bar{x}_n) + (1 - y_n) \ln(1 - \sigma(w^T \bar{x}_n))$$

finding the min by setting derivative to 0

$$
\begin{aligned}
\frac{dL}{dw} &= 0 \\
0 &= \sum_n [\sigma(w^T \bar{x}_n) - y_n] \bar{x}_n
\end{aligned}
$$

and since we can't isolate w we use **Newton's Method** to iteratively solve:

$$w \to w - H^{-1} \nabla L(w)$$

where $H = \bar{X}R\bar{X}^T$ is the hessian

$$R = \begin{bmatrix} \sigma_1(1-\sigma_1) & & \\ & ... & \\ & & \sigma_N(1-\sigma_N) \end{bmatrix}$$

$$\sigma_k = \sigma(w^T \bar{x}_k)$$

6.3 Regularization

To ensure that we can inverse H (so it isn't singular), add λ:

$$H = \bar{X}R\bar{X}^T + \lambda I$$

6.4 Non-linear Regression

Non-linear regression using the same algorithm, map inputs to a different space!
Use non-linear basis functions ϕ_i, so for

$\phi_0(x) = 1$
$\phi_1(x) = x$
$\phi_2(x) = x^2$

the hypothesis space is $H = \{x \leftarrow w_0\phi_0(x) + w_1\phi_1(x) + w_2\phi_2(x) | w_i \in \Re\}$

common basis functions:

- polynomial $\phi_j(x) = x^j$
- gaussian $\phi_j(x) = e^{-\frac{(x-\mu_j)^2}{2s^2}}$
- sigmoid $\phi_j(x) = \sigma(\frac{x-\mu_j}{s})$
- fourier, wavelets ...

7 Perceptron

7.1 Computer vs Brain

Computer:

- bunch of gates
- electrical signals by gates
- sequential and parallel
- fragile

Brain

- network of neurons

- nerve signal propogate
- parallel
- robust (neurons die)

ANN Unit consists of weights w and activation function h, so that output $y_j = h(W_j\bar{x})$. Structure is either **feed-forward** or **recurrent**

7.2 Perceptron Learning

Learning is done separately for each unit j:

for all pairs (x, y) **do**
  if output is correct **then**
    Do nothing
  else if output $= 0$, label $= 1$ **then**
    $\forall_i W_{ji} \to W_{ji} + x_i$
  else if output $= 1$, label $= 0$ **then**
    $\forall_i W_{ji} \to W_{ji} - x_i$
  end if
end for

7.3 Alternative Learning

let M be the set of misclassified examples (where $y_n w^T \bar{x}_n < 0$) then find w to minimize number of misclassifications:

$$E(w) = -\sum_{(x_n, y_n) \in M} y_n w^T \bar{x}_n$$

Use **gradient descent**

$$w \leftarrow w - \eta \nabla E$$

where η is the learning rate

If we adjust w one example at a time, we use **sequential gradient descent** which is equivalent to threshold perceptron learning when $\eta = 1$

7.4 Linear Separability

Threshold perceptron converges iff the data is linearly separable

linear separator $w^T \bar{x}$, since it is linear in w

7.5 Other Networks

Sigmoid Perceptron "soft" linear separators (same H as linear regression)

$$E(w) = \frac{1}{2} \sum_n (y_n - \sigma(w^T \bar{x}_n))^2$$

for all (x_n, y_n) **do**
  $E_n \leftarrow y_n - \sigma(w^T \bar{x}_n)$
  $w \leftarrow w + \eta E_n \sigma(w^T \bar{x}_n)(1 - \sigma(w^T \bar{x}_n))\bar{x}_n$
end for

8 Multilayer Neural Networks

8.1 Introduction

Previously, our basis functions were fixed, but we can remove that restriction by learning non-linear basis functions.

$$\text{hidden unit } z_j = h_1(w_j^{(1)} \bar{x})$$
$$\text{output unit } y_k = h_1(w_j^{(2)} \bar{x})$$
$$\text{both units } y_k = h_2(\sum_j w_{kj}^{(2)} h_1(\sum_i w_{ji}^{(1)} x_i))$$

if we consider our hidden input to be a basis function, then this is equivalent to a linear regression and a learned basis function, e.g.

- non-linear regression: h_1 is a non-linear function, h_2 is identity
- non-linear classification: h_1 is a non-linear function, h_2 is sigmoid

8.2 Backpropogation

Error function:

$$E(w) = \frac{1}{2} \sum_n \|f(x_n, W) - y_n\|_2^2$$
$$\text{where } f(x, W) = \sum_j w_{kj}^{(2)} \sigma(\sum_i w_{ji}^{(1)} x_i)$$

so our update rule for gradient descent is

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\delta E_n}{\delta w_{ji}}$$
$$\text{where } \frac{\delta E_n}{\delta w_{ji}} = \delta_j z_i$$

Do gradient update in two phases:

1. **forward**: compute output z_j of each unit j

$$z_j = h(\sum_i w_{ji} z_i)$$

2. **backward**: compute delta δ_j at each unit j

$$d_j = \begin{cases} h'(a_j)(z_i - y_i), & \text{if } j \text{ is output} \\ h'(a_j) \sum_k w_{kj} \delta_k & \text{if } j \text{ is a hidden unit before } k \end{cases}$$

Analysis:

- fast computation
- slow convergence, may get trapped in local optima
- prone to overfitting, solve with
  - early stopping
  - regularization

9 Kernel Methods

9.1 Introduction

Data may not be linearly separable, so map into a high-dimensional space where it is! This is computationally difficult though, so instead calculate a similarity measure (dot product) in the high dimensional space and use algorithms that only need that measure.

kernel methods use large set of fixed non-linear basis functions, with a "dual trick" to make complexity depend on amount of data instead of number of basis functions

kernel function $k(x, x') = \phi(x)^T \phi(x')$ for some basis function $\phi(x)$

linear regression objective, setting derivative to 0 gives us

$$w = -\frac{1}{\lambda} \sum_n (w^T \phi(x_n) - y_n) \phi(x_n)$$

so w is a linear combination of inputs in feature space

$$\begin{aligned}
&= \{\phi(x_n) | 1 \leq n \leq N\} \\
\text{substitute } w &= \phi a \\
\text{where } \phi &= [\phi(x_1) \dots \phi(x_n)] \\
a &= [a_1, \dots a_n]^T
\end{aligned}$$

now let $K = \Phi^T \Phi$, therefore our prediction

$$\begin{aligned}
y_* &= \phi(x_*)^T \Phi a \\
&= k(x_*, X)(K + \lambda I)^{-1} y
\end{aligned}$$

For this we need to just find dual solution a instead of w

- now depends on # of data instead of # of basis function
- can use many more basis functions
- don't actually need Φ, just need a semi-definite kernel $K, \exists \Phi \mid K = \Phi^T \Phi$ or all eigenvalues ≥ 0

9.2 Common Kernels

Common kernels:

- polynomial $k(x,x') = (x^T x' + c)^M, c \geq 0$
- gaussian $k(x,x') = \exp(-\frac{\|x-x'\|^2}{2\sigma^2})$

Also construct more kernels using rules. Let $k_1(x,x')$ and $k_2(x,x')$ be valid kernels, and $x = \begin{pmatrix} x_a \\ x_b \end{pmatrix}$ then it is also valid $k(x,x') =$

- $ck_1(x,x')\ \forall c > 0$
- $f(x)k_1(x,x')f(x')\ \forall f$
- $q(k_1(x,x'))$ where q is a polynomial with coeffs ≥ 0
- $\exp k_1(x,x')f(x')$
- $k_1(x,x') + k_2(x,x')$
- $k_1(x,x')k_2(x,x')$
- $k_3(\phi(x), \phi(x'))$
- $x^T Ax'$ where A is symmetric positive semi-definite
- $k_a(x_a, x'_a) + k_b(x_b, x'_b)$
- $k_a(x_a, x'_a)k_b(x_b, x'_b)$

Kernels can also be defined wrt to sets, strings, graphs. E.g. $k(d_1, d_2) =$ similarity between two documents

9.3 Example

Show that $k(x,z) = (x^T z)^2$ is a valid kernel by finding ϕ

$$\begin{aligned} k(x,z) &= (x^T z)^2 \\ &= (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2}x_1 x_2, x_2^2)(z_1^2, \sqrt{2}z_1 x_2, z_2^2)^T \\ &= \phi(x)^T \phi(z) \end{aligned}$$

after separating x and z we find that

$$\phi(x) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)^T$$

10 Gaussian Processes

10.1 Overview

We want to do regression even when we don't know the exact type of function. We assume that $p(f(x))$ is a gaussian for all points x, and use a kernel function as the distance between any two points. Then we can find a general function f

$$\begin{aligned} f(x) &= GP(\text{mean}, \text{kernel covariance}) \\ m(x) &= E(f(x)) \\ k(x, x') &= E((f(x) - m(x))(f(x') - m(x'))) \end{aligned}$$

10.2 Gaussian Process Regression

$$\begin{aligned} \text{prior } p(f(x)) \quad &= N(m(x), k(x, x)) \\ \text{likelihood } p(y|X, f) \quad &= N(f(x), \sigma^2 I) \\ \text{posterior } p(f(x)|X, y) &= N(\bar{f}(x), k'(x, x)) \\ \text{where } \bar{f}(x) \quad &= k(x, X)(K + \sigma^2 I)^{-1} y \\ k'(x, x) \quad &= k(x, x) - k(x, X)(K + \sigma^2 I)^{-1} k(X, x) \\ \text{prediction } p(y|x, X, y) \quad &= N(\bar{f}(x), k'(x, x)) \end{aligned}$$

Inversion of $K + \sigma I$ (and therefore this process) is cubic in number of training points (vs bayesian linear regression which is cubic in number of *basis functions*

11 Support Vector Machines

11.1 Overview

Find a separator such that it has maximum margins with nearest data

$$\max_x \frac{1}{\|w\|} (\min_n y_n w^T \phi(x_n))$$

Fix the minimal distance to 1, minimize $\|w\|$

$$\begin{aligned} &\min_w \frac{1}{2} \|w\|^2 \\ &\text{s.t. } y_n w^T \phi(x_n) \geq 1 \, \forall n \end{aligned}$$

The points that decide our separator, where $y_n w^T \phi(x_n) = 1$, are known as the **support vectors**

11.2 Dual

Reformulate as penalty for x_n violating constraint

$$\max_{a>0} \min_{w} L(w,a)$$
$$\text{where } L(w,a) = \frac{1}{2}\|w\|^2 - \sum_n a_n(y_n w^T \phi(x_n) - 1)$$

substitute $w = \sum_n a_n y_n \phi(x_n)$

$$\max_{a} L(a) \text{ s.t. } a_n \geq 0$$
$$\text{where } L(a) = \sum_n a_n - \frac{1}{2}\sum_n \sum_{n'} a_n a_{n'} y_n y_{n'} k(x_n, x_{n'})$$

It can be sparse (many a_ns are 0) and also classification only needs the kernel

$$y_* = sign(\sum_n a_n y_n k(x_n, x_*))$$

11.3 Comparison to Perceptron

Perceptron:

- linear separator
- simple update rule
- prone to overfitting

SVM:

- unique max-margin linear separator
- quadratic optimization
- robust to overfitting

11.4 Soft Margins

If the data is not linearly separable, relax constraints to allow for misclassification:

$$y_n w^T \phi(x_n) \geq 1 - \epsilon_n, \epsilon_n \geq 0 \quad \forall n$$

Objective is to minimize the misclassification (as denoted by ϵ)

$$\min_{w,\epsilon} C \sum_{n=1}^{N} \epsilon_n + \frac{1}{2}\|w\|^2$$

- C acts as the tradeoff between error minimization and model complexity
- $C \to \infty$ gives the hard margin
- can handle minor misclassifications but still sensitive to outliers
- all points on the margin, inside the margin, or misclassified are support vectors

11.5 Mulitclass SVM

11.5.1 One-against-all

Train k SVMs to distinguish each class from the rest, but two classes might both say the example belongs to them

11.5.2 Pairwise Comparison

Train $O(k^2)$ SVMs to compare between each pair of classes

11.5.3 Continuous Ranking

Single SVM that returns a continuous value to rank all classes

$$y_* = argmax_k w_k^T \phi(x_*)$$

Multi-class margin:

for each class $k \neq y$ define a linear constant

$$w_y^T \phi(x) - w_k^T \phi(x) \geq 1$$

therefore the optimization is

$$\min_w \frac{1}{2} \sum_k \|w_k\|^2$$
$$\text{s.t. } w_{y_n}^T \phi(x_n) - w_k^T \phi(x_n) \geq 1 \ \forall n, k \neq y_n$$

12 Hidden Markov Model

12.1 Assumptions

1. stationary process: transition and emission is the same at each step

$$Pr(x_t|y_t) = Pr(x_{t+1}|y_{t+1}) \ \forall t$$
$$Pr(y_t|y_{t-1}) = Pr(y_{t+1}|y_t) \ \forall t$$

2. markovian process: next state only depends on previous

$$Pr(y_{t+1}|y_t, y_{t-1} \dots y_1) = Pr(y_{t+1}|y_t) \ \forall t$$

12.2 Inference

- monitoring $Pr(y_t|x_{1 \dots t})$
- prediction $Pr(y_{t+k}|x_{1 \dots t})$
- hindsight $Pr(y_k|x_{1 \dots t})$ where $k < t$
- most likely estimation $argmax_{y_1 \dots y_t} Pr(y_{1 \dots t}|x_{1 \dots t})$

12.2.1 Monitoring

$$P(y_t|x_{1...t}) = Pr(x_t|y_t)\sum_{y_{t-1}} Pr(y_t|y_{t-1})Pr(y_{t-1}|x_{1...t-1})$$

1. Compute $Pr(y_t|x_{1...t})$ with forward algorithm:

$Pr(y_1|x_1) \propto Pr(x_1|y_1)Pr(y_1)$
for $i = 1 \to k$ **do**
  $Pr(y_i|x_{1...i}) \propto Pr(x_i|y_i)\sum_{y_{i-1}} Pr(y_i|y_{i-1})Pr(y_{i-1}|x_{1...i-1})$
end for

12.2.2 Prediction

$$Pr(y_{t+k}|x_{1...t}) = \sum_{y_{t+k-1}} Pr(y_{t+k}|y_{t+k-1})Pr(y_{t+k-1}|x_{1...t})$$

1. Compute $Pr(y_t|x_{1...t})$ with forward algorithm as previously
2. Compute $Pr(y_{t+k}|x_{1...t})$ with forward algorithm:

for $i = 1 \to k$ **do**
  $Pr(y_{t+i}|x_{1...t}) = \sum_{y_{i-1}} Pr(y_{t+i}|y_{t+i-1})Pr(y_{t+i-1}|x_{1...t})$
end for

12.2.3 Hindsight

$$Pr(y_k|x_{1...t}) = Pr(y_k|x_{1...k})Pr(x_{k+1...t}|y_k)$$

1. Compute $Pr(y_t|x_{1...t})$ with forward algorithm as previously
2. Compute $Pr(x_{k+1...t}|y_k)$ with backwards algorithm:
   for $i = t-1 \to k$ **do**
     $Pr(x_{i...t}|y_{i-1}) = \sum_{y_i} Pr(y_i|y_{i-1})Pr(x_i|y_i)Pr(x_{i+1...t}|y_i)$
   end for
3. Compute $Pr(y_k|x_{k+1...t}) = Pr(x_{k+1...t}|y_k)$

12.2.4 Most Likely Explanantion

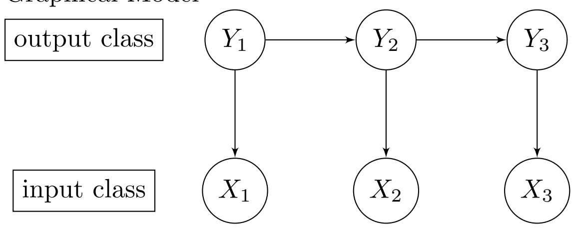
Most likely sequence of events (classes) given measuresments

$$argmax_{y_1...t}Pr(y_{1...t}|x_{1...t})$$
$$\max_{y_{1...t}} Pr(y_{1...t}|x_{1...t}) = \max_{y_t} Pr(x_t|y_t) \max_{y_{1...t-1}} Pr(y_{1...t}|x_{1...t-1})$$

Compute $\max_{y_{1...t}} Pr(y_{1...t}|x_{1...t})$ with dynamic programming, **Viterbi Algorithm**
$Pr(y_{1...2}|x_1) \propto \max_{y_1} Pr(y_2|y_1)Pr(x_1|y_1)Pr(y_1)$
for $i = 2 \to t-1$ **do**
  $Pr(y_{1...i+1}|x_{1...i}) \propto \max_{y_i} Pr(y_{i+1}|y_i)Pr(x_i|y_i)\max_{y_{1...i-1}} Pr(y_{1...i}|x_{1...i-1})$
end for
$Pr(y_{1...t}|x_{1...t}) \propto \max_{y_t} Pr(x_t|y_t)\max_{y_{1...t-1}} Pr(y_{1...t}|x_{1...t-1})$

12.3 Parametrization

Graphical Model



Parametrized distributions:

- transition $P(Y_t|Y_{t-1}) = Mulitnomial_{Y_{t-1}}$
- emission $P(X_t|Y_t) = Gaussian(\mu_y, \sigma_y)$
- start $P(Y_1) = Multinomial$
- joint $P(X_{1:k}, Y_{1:k}) = P(Y_1) \prod_{t=1}^{k} P(X_t|Y_t) \prod_{t>1}^{k} P(y_t|y_{t-1})$

12.4 Supervised Learning

12.4.1 Multinomial Emissions

Objective:

$$argmax_{\pi,\theta,\phi} Pr(y_{1\ldots t}, x_{1\ldots t}|\pi, \theta, \phi)$$

For $y \in \{c_1, c_2\}, x \in \{v_1, v_2\}$, maximum likelihood:

$$\begin{aligned}
\pi_{c_1^{\text{start}}} &= \#c_1^{\text{start}}/(c_1^{\text{start}} + c_2^{\text{start}}) \\
\theta_{c1|c1} &= \#(c_1, c_1)/(\#(c_1, c_1) + \#(c_2, c_1)) \\
\theta_{c1|c2} &= \#(c_1, c_2)/(\#(c_1, c_2) + \#(c_2, c_2)) \\
\phi_{v1|c1} &= \#(v_1, c_1)/(\#(v_1, c_1) + \#(v_2, c_1)) \\
\phi_{v1|c2} &= \#(v_1, c_2)/(\#(v_1, c_2) + \#(v_2, c_2))
\end{aligned}$$

12.4.2 Gaussian Emissions

$$\begin{aligned}
\pi_{c_1^{\text{start}}} &= \#c_1^{\text{start}}/(c_1^{\text{start}} + c_2^{\text{start}}) \\
\theta_{c1|c1} &= \#(c_1, c_1)/(\#(c_1, c_1) + \#(c_2, c_1)) \\
\theta_{c1|c2} &= \#(c_1, c_2)/(\#(c_1, c_2) + \#(c_2, c_2)) \\
\mu_{c1} &= \text{mean of } c_1 \\
\mu_{c2} &= \text{mean of } c_2 \\
\sigma_{c1}^2 &= \frac{1}{\#c_1} \sum_{t|y_t=c_1} (x_t - \mu_{c_1})^2 \\
\sigma_{c2}^2 &= \frac{1}{\#c_2} \sum_{t|y_t=c_2} (x_t - \mu_{c_2})^2
\end{aligned}$$

12.4.3 Monitoring and MLE

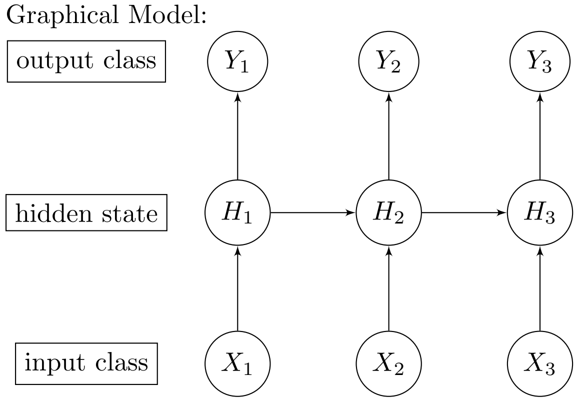
Monitoring: what is the probability of $y_i = c_1$ at any i?
iterate $Pr(y_i|x_{1...i}) \propto Pr(x_i|y_i) \sum_{y_{i-1}} Pr(y_i|y_{i-1})Pr(y_{i-1}|x_{1...i-1})$

Most Likely Explanation: $argmax_{y_1...y_t} Pr(y_{1...t}|x_{1...t})$? Viterbi algorithm!
iterate $\max_{y_{1...i}} Pr(y_{1...i+1}|x_{1...i}) \propto \max_{y_i} Pr(y_{i+1}|y_i)Pr(x_i|y_i) \max_{y_{1...i-1}} Pr(y_{1...i}|x_{1...i-1})$

Example 12.1. Belief Monitoring and Most Likely Sequence see CS489-HMM.pdf

13 Recurrent Neural Networks

Graphical Model:



HMMs have more flexible queries, RNNs have more expressive distributions

13.1 Belief Monitoring

$P(Y_1, X_1) =$

- sigmoid or softmax (distribution over binary and categorial outputs)
- linear (gaussian distribution with unit variance)

14 Deep Neural Networks

14.1 Overview

Deep Neural Network ann with many hidden layers

\+ high expressiveness

\- vanishing gradient

\- avoiding overfitting

Applications:

- speech recognition
- image recognition: ImageNet (3.07% by Inception V4)

- machine translation
- control ...

14.2 Vanishing Gradient

With sigmoid and hyperbolic activation function, gradient is always *leq* 1. Therefore, since backprop at each layers multiplies by gradient, the gradient becomes very small as we progress back through layers

$$\frac{\delta y}{\delta w_4} = \sigma'(a_4)\sigma(a_3)$$
$$\frac{\delta y}{\delta w_3} = \sigma'(a_4)w_4\sigma'(a_3)\sigma(a_2) \leq \frac{\delta y}{\delta w_4}$$

Solutions:

- pre-training
- ReLU $h(a) = \max(0, a)$
  - gradient is 0 or 1
  - sparse computation
  - softplus $h(a) = \log(1 + e^a)$
- Maxout: learn a piecewise activation function that is the max of k linear functions (used with dropout)
  - $h_i(x) = \max_{j \in [1,k]}(z_{ij})$ where $z_{ij} = x^T W_{\dots ij} + b_{ij}$
  - doesn't have any places where gradient is close to 0
  - enhances accuracy of dropout averaging

14.3 Overfitting

High expressivity increases risk of overfitting
Solutions:

- regularization
- data augmentation
- dropout: randomly drop some units from the network when training
  - training: unit x is dropped with prob p
  - testing: multiply output of unit x by $1 - p$
  - approximate form of ensemble learning

15 Convolutional Neural Networks

15.1 Convolutions

continuous

$$
\begin{aligned}
y(i) &= \int_t x(t)w(i-t)dt \\
&= (x * w)(t)
\end{aligned}
$$

discrete

$$
y(i) = \sum_{-\infty}^{\infty} x(t)w(i-t)dt
$$

Example 15.1. vertical edge detection

$$
\begin{aligned}
y(i,j) &= x(i,j) - x(i-1,j) \\
w(i-t_1, j-t_2) &= \begin{cases} 1, & \text{if } t_1 = i, t_2 = j \\ -1, & \text{if } t_1 = i+1, t_2 = j \\ 1, & \text{otherwise} \end{cases}
\end{aligned}
$$

15.2 CNN

In neural networks

convolution linear combination of a subset of units based on a specific pattern of weights

pooling commutative math operation combining several units (max, min, average ...)

CNN network with alternating convolution and pooling, where some convolution weights are shared

$$
a_j = \sum_i w_{ji} z_i
$$

combine with an activation function to produce a feature

$$
z_j = h(\sum_i w_{ji} z_i)
$$

Benefits:

- sparse interactions
- parameter sharing
- locally equivarient

- locally invariant to translation
- handle inputs of varying length

Applications:

- image processing
- data with sequential, spatial, or tensor patterns

16 Recurrent and Recursive Neural Networks

16.1 Recurrent Neural Network

Output is fed back to the network as inputs, so the network can unroll to handle variable length data

- train with BPTT, by unrolling the network a number of steps
- weight sharing combines gradients of shared weights into single gradient
- challenges
  - gradient vanishing and exploding
  - long range memory
  - prediction drift

Long Short Term Memory (LSTM) networks tackle that challenge by using gates to control memory (input, forget, output)

16.1.1 Uses

- belief monitoring
  - RNN can simulate and generalize HMM
- bi-directional RNN
  - combine past and future evidence
- encoder-decoder
  - machine translation
  - question answering
  - dialog

16.2 Recursive Neural Network

Generalize RNN from chains to trees

- weight sharing allows trees to fit variable length data
- useful for graphs, e.g. semantic parsing tree

17 Autoencoders

17.1 Overview

Feed-forward neural network for

- compression
- denoising
- sparse representation
- data generation

$$
\begin{aligned}
f(x) &= z \\
g(z) &= x
\end{aligned}
$$

17.2 Linearity

17.2.1 Linear

f and g are both linear

- hidden node is a compressed representation
- equivalent to PCA if using euclidean norm (squared loss)

$$\min_W \frac{1}{2} \sum_n \|W_g W_f x_n - x_n\|_2^2$$

17.2.2 Non-Linear

f and g are both non-linear

- hidden node is a non-linear manifold

$$\min_W \frac{1}{2} \sum_n \|g(f(x_n; W_f); W_g) - x_n\|_2^2$$

17.3 Deep and Sparse

In theory, one hidden layer is sufficient to represent any possible compression, but in practise more layers is better. When there are more hidden nodes than inputs, use regularization to constrain: e.g. sparsity with a penalty term

$$
\begin{aligned}
&\min_W \frac{1}{2} \sum_n \|g(f(x_n; W_f); W_g) - x_n\|_2^2 + c\delta(f(x_n; W_f)) \\
&\text{where } \delta(f(x_n; W_f)) = \begin{cases} 1, & \text{if } f(x_n; W_f) \neq 0 \\ 0, & \text{if } f(x_n; W_f) = 0 \end{cases}
\end{aligned}
$$

approximate objective is L1 regularization

$$\min_W \frac{1}{2} \sum_n \|g(f(x_n; W_f); W_g) - x_n\|_2^2 + c \|f(x_n; W_f)\|_1$$

17.4 Denoising

Consider $\tilde{x}$, the noisy version of input x. Use autoencoder to compress to true representation and reconstruct without noise

$$\min_W \frac{1}{2} \sum_n \|g(f(\tilde{x}_n; W_f); W_g) - x_n\|_2^2 + c \|f(\tilde{x}_n; W_f)\|_1$$

17.5 Probabilistic

Let f and g represent conditional distributions

$$f = Pr(h|x; W_f)$$
$$g = Pr(x|h; W_g)$$

by using sigmoid, softmax, or linear units at the hidden and output layers