Machine Learning MU 2015

Barak A. Pearlmutter   Machine Learning - Class of 2016

Contents

Lecture 01

CS401 Machine Learning & Neural Networks

Three main types of Machine Learning

- Supervised Learning (regression)
- Unsupervised Learning (figure out structure, data mining)
- Reinforcement Learning

Wellsprings of Machine Learning

- Making computers do things (automatic programming)
- Statistics (making inferences from data)
- Neuroscience and psychology and cognitive science
- Information theory
- Physics (especially thermodynamics)

Think of a baby, that learns and takes in data constantly

In the beginning

In the dawn of the computer age, computers had poor languages, severe limitations, and high hopes and ambitions. An example problem was translating Russian to English or vice versa. People sat down with Russian-English dictionaries and attempted to write a program that would do it. Another example would be telling a computer to identify a tank in a photograph, or even translate a scanned page of text from a bitmap to text. People are good at these jobs, and computers are not.

An example to think of would be a big box, with lots and lots of knobs on it. These knobs don't have set positions; they rotate freely. If people were to spend years trying to get these knobs in the correct position, such that they do something we want, they may never get it to work. Machine learning would enable us to do it automatically.

Supervised Learning

This is supervised learning, or regression: the task of inferring a function from labelled training data.

Example: Tank recognition in an image

data

input	desired output
image 1	yes
...	...
image M	no

Image → [Machine] → Classification, class label, or probability

How does regression work?

If we call each image a vector (that is, 3 2x2 RGB arrays), then

inputs → desired outputs

x(i) → y(i)

We can call all the settings of all the knobs on our theoretical machine w, with w being a vector (or data structure in code).

x → Machine → ŷ

ŷ = f(x;w)

E = (1/m) sum_i = 0…(m-1) (1/2) || f(x(i);w) - y(i) ||ˆ2

Linear Regression

f(x;w) = W x

This is linear regression, a special case of regression analysis, which tries to explain the relationship between a dependent variable and one or more explanatory variables.

Examples of machine learning in everyday applications include

- predicting housing prices
- language translation
- face detection in cameras
- voice recognition

Unsupervised Learning

Another form of machine learning is unsupervised learning, the objective being to try to find hidden structure in unlabelled data. Since the examples given to the machine are unlabelled, there is no error or reward signal to evaluate a potential solution.

Clouds (i.e., Amazon, Google, etc.) devote a lot more cycles to unsupervised learning than to other forms of machine learning, as a lot of unlabelled data is freely available to them, via an internal database, via data collected by a spider trawling the web, or even via shopping trends.

Reinforcement Learning

The third, and final type of machine learning is reinforcement learning. This is the most "biological" of the types, inspired by behaviourist psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. That is to say, the machine is put into an environment and is given "rewards" for getting things right.

Reinforcement learning subtypes:

- Without time ("bandit" problem - the problem a gambler faces at a row of slot machines ("one-armed bandits"), when deciding which machines to play, how many times to play each machine and in which order to play them.)

- With time (credit assignment - there are a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment. It is required to perform all tasks by assigning exactly one agent to each task and exactly one task to each agent in such a way that the total cost of the assignment is minimized.)

Hybrid

There exist hybrid systems, which combine two or more of the three above. An example of a hybrid system would be IBM's translation system attempts early on. They treated language translation as a machine learning problem, trying to learn the properties of a communication channel and convert it. To translate from French to English and vice versa they used French parliamentary proceedings as their data set, as they are made available in both languages.

Assignment:

- Soon we'll get an assignment.
- Grade each other's (double blind) to make a sparse matrix.
- Use this as a dataset for some interesting problems.

Lecture 02

Types of Machine Learning

1. Supervised
2. Unsupervised
3. Hybrid-RL

Perceptron fits under supervised.

Perceptron

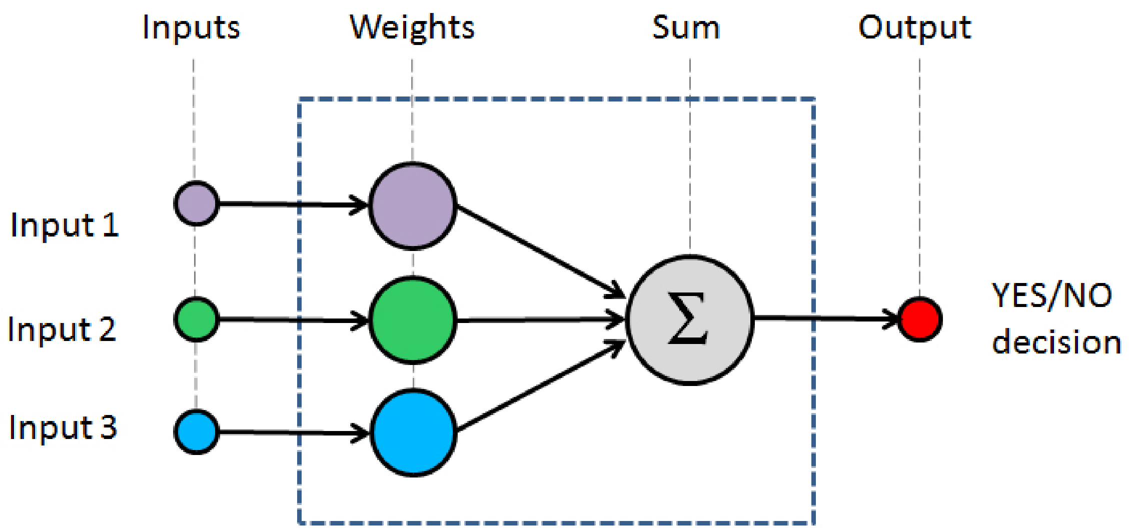


Figure 1: Picture of perceptron

The Perceptron is made up of 3 main components: Input xi (can be many dimensions, 2D - yes/no, higher - what's in a picture? dog, cat? etc.)

Output: $\hat{y}$ (what the machine spits out)

Correct: y(i) (what the machine should say)

Dials/knobs - weights, w(i) is a vector of numbers usually but could be a data structure, or many separate vectors but we'll treat it as one

You can turn and fiddle with the dials to make things nice, usually by some objective function. Try hard to get the correct answer - non-objective, this is what it used to be, they used rule of thumb.

Perceptron is an early ML algorithm which fiddles with the dials to get the output correct for a given input. If you do this repeatedly for a data set, you can prove then it will get the correct answer on all of them.

Usually we have some error measure, usually a fraction of what it gets right, we'll call this E, we want this to be low.

E = error over entire training set

E = 1/m((i=0, m-1) E(i)) sum over all cases or average, where m is the number of inputs = 1/m((i) E(i)) = < (i)> average over i

Construction of Perceptron

Outputs are binary; 3 options: **True/False**, **1/0**, **+1/-1**

1/0 useful if you expect lots of zeros e.g. hand-written digit recognition

+1/-1 useful if you're expecting some sort of symmetry and can use this to figure out if it was right or wrong

w(j) will be our voltage and is our threshold so we end up with equation: $\hat{y}$ = (j) x(j) * w(j) >

Basically the inputs x(j) are sent in and run through wires going through resistor which are controlled by the dials, these are then added up and the machine then checks if this is greater than the threshold.

Not quite what we want, we want them all to be a whole function

so $\hat{y}$ = sign((j=1, n) w(j)*x(j) -) this is the Transfer function of a Perceptron. This equation will return -1 if $\hat{y}$ is less than the threshold and +1 if it's greater than it.

So what was the motivation behind the Perceptron? ## Neurons

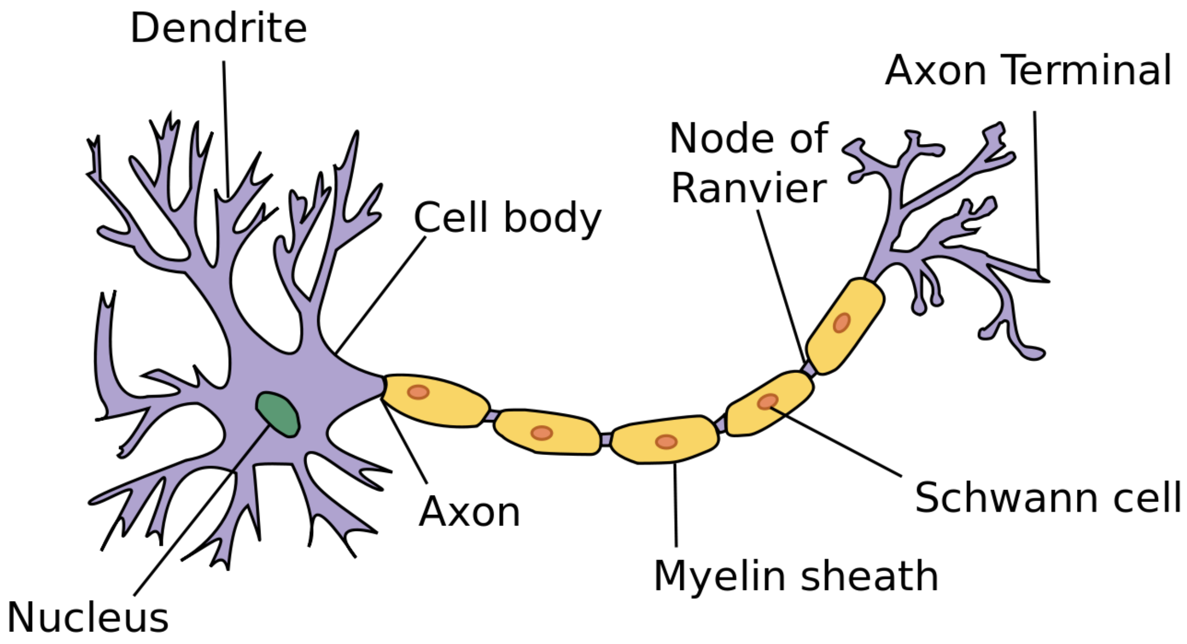


Figure 2: Neuron diagram

In the late 1940's, it was figured out how neurons work. Alan Hodgkin and Andrew Huxley performed experiments on the giant squid axon, recording ionic currents (for which they got the 1963 Nobel Prize in Physiology and Medicine)

The potential difference between the inside of a neuron and the outside is -80mV. A pulse is transmitted through the axon terminal, causing a chemical reaction and the "gate opens up". This allows some particles through - ions. Sodium rushes into the neuron, and the voltage increases past the threshold - this is a spike. Now other neurons can figure out what the neuron is doing.

Neurons have “amplifiers” on them called axons, this is because if the signal was too small it would die out and if it was too big it would keep amplifying.

In a neuron, the maximum spiking rate is 1kHz. This rate is only seen in a dying neuron though. The average spiking rate is ~0.1Hz.

Why don't we use chemical reactions like this in computers? Cause its slow!

How to adjust the weights on the machine?

We can change 2 things, the weight or the threshold. For our example we'll set theta = -w(0) and x(0) = 1

ŷ = sign((j=0, n) w(i)*x(i)) (we'll call the bit in brackets z i.e. z = (j=0, n) w(j)*x(j))

Imagine we have a LEARN button on the machine, if this is pressed it can do 2 things: If ŷ and y are the same, do nothing. Otherwise adjust the weights accordingly.

ŷ	y	do
+1	+1	nothing
-1	-1	nothing
-1	+1	change w to make z > 0
+1	-1	change w to make z < 0

In modern techniques you make very small changes to the weights over and over, until you get the desired output. With the Perceptron it would make a big adjustment, so that it got the right answer, but the smallest change possible to get this result.

But how can we do this? Use Linear Algebra. Think of it as symmetrically pushing vectors around.

Input space - x R(n) (this is the space the inputs of the machine can “live” in)

In our case it will be a 2D input but it could be more (video, text, audio etc.)

Aside - ours is actually 3 as we set xo = 1, but we can ignore this for maths purposes

Weight space - w R(n)

Output space - binary (yes/no or in our case +1 or -1)

How are we going to change it so that we get the correct response? We've 2 options of what to look at:

For a particular w we look at the input (x), break it up to which we give our plus or minus 1.

For a particular x we look at weights (w) and see which give plus or minus 1.

In this case w is a separating surface, and we need to turn w. In our graph we'd want to turn it clockwise, but what about in general? It's not always 2D.

We want to move w towards x.

ŷ	y	do
+1	+1	nothing
-1	-1	nothing
-1	+1	change w to make z > 0 w(t+1) = w(t) + x
+1	-1	change w to make z < 0 w(t+1) = w(t) - x

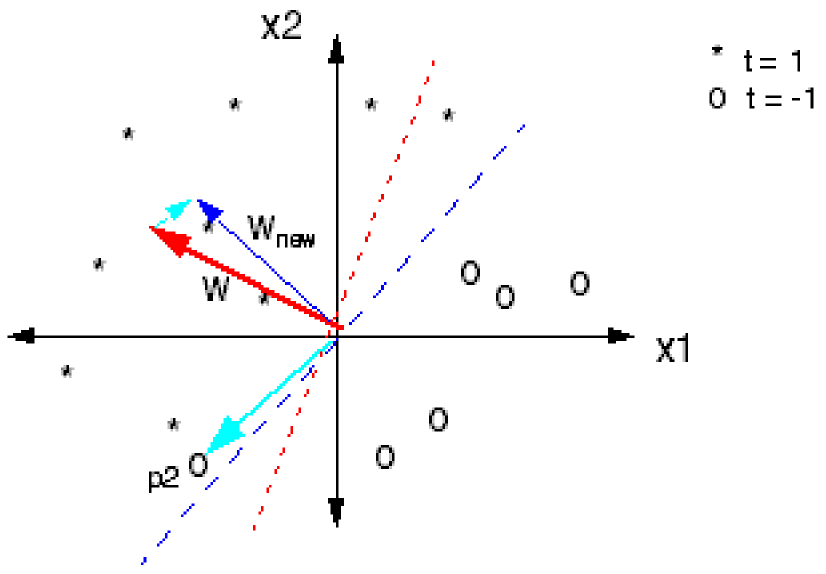


Figure 3: perceptPict-2

w̃(t+1) = w(t) - *x

w(t+1) = w̃(t+1)/w̃(t+1)

We want to solve: (w + x) . x = 0 (i.e. solve for 0)
(the dots on this and the next few lines represent dot products)

w.x + *x.x = 0

w.x + ||x||ˆ2 = 0

 = w.x/||x||ˆ2 (this would be just on the separation plane)

 = - w.x/||x||ˆ2 + d (the plus d ensures it's just over the separation plane)

To make it work for both cases:

 = (-w.x/||x||ˆ2 + d) * y

 = (-w.x/||x||ˆ2 + d) * (y - ŷ/2) (this accounts for the do nothing case

What ? Should we normalise w? Turns out there's not much difference, convenient not too big or small.

If the output is wrong, press learn and it'll be right next time. If it sees all inputs, it'll get all of the outputs right.

Aiming for the smallest tilt to get it right. If there's no setting to get it right, then it won't work. Assignment —

Machine Learning in the News

Find a story about machine learning in the popular press (broadly construed, to include blogs or whatever) and write a summary of the story, from a technical perspective, about 1/2 page in length. Information to include would be things like what the problem they're trying to solve is, how they addressed it, where they got their data, how much data they had, and what it looked like, why they considered the problem interesting, etc. Turn it in as a plain text (or simple markdown format) file. No figures or diagrams or equations—we'll be doing machine learning with these as a dataset at that would complicate our lives. Also, be sure to do a spelling correction, because misspellings will make our lives more difficult later. Do not include your name in the file; I can get that from Moodle, and it would be a hassle to strip names off. Remember that other students will be marking these (in addition to myself) so please try to give them something interesting to read; something you'd enjoy reading yourself.

Due: Mon 28-Sep-2015.

(I will process them before class Tuesday, so please have them all in by 2pm Monday because it is a hassle to slip late ones into the processing pipeline.)

Notes from previous years

NOTE: Pictures wouldn't cross over so only text

Perceptron Learning Rule

Input comes in Checks if correctly classified if it is → do nothing if it's not → change weights by the minimum amount (rotate w a little bit) so that it would be correctly classified

Convergence rule: if w exists, then by cycling through the inputs you'll eventually find w, where all inputs are classified correctly.

w doesn't exist if there is even a single outlier. No admissible solution.

Perceptron: • Invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt. • Funded by the United States Office of Naval Research. Used to distinguish tanks from their surrounding environment. • This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors. • It was later realized that the perceptron was influenced not only by the shapes of images given for its interpretation but it also was effected by the brightness and was unable to clarify the presence of tanks when there was a different brightness to the time the tank present data was taken. How does the perceptron work?

Figure 1.: is a graphical illustration of a perceptron with inputs, ..., and output (sourced from http://reference.wolfram.com/applications/neuralnetworks/NeuralNetworkTheory/2.4.0.html) As seen in figure 1 the weighted sum of the inputs and the unity bias are first summed and followed by being processed by a step function yielding the output (x, w, b) = UnitStep (w1 x1 + w¬2 x¬2 + ... + wn xn + b) Where {w1. . . wn} are the weights applied to the input vector and b is the bias weight. Each of the weights are represented by the arrows in figure 1. The UnitStep function is 0 for arguments less than 0 and 1 elsewhere. So can take values of 0 or 1 depending on the value of the weighted sum. The perceptron can indicate 2 classes corresponding to these 2 input values. While in the training process, the weights (inputs and bias) are adjusted so the input data is mapped correctly to one of the two classes. Off sample performance more important!!!

Cross validation Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. Generally better for a scenario where the goals are predicted, and the test is to see how accurate the prediction is by having a training set and a validation set. The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent data set (i.e., an unknown dataset, for instance from a real problem). A bigger data set presumably gives a better generalization! figure 1: the line between +'s and −'s is constantly moving until it fits to a particular position where the margin between the −'s and +'s are equal. A problem with this will be if they aren't separated appropriately. 3 common problems causing this may be outliers, noise and mislabelled points. It is best to keep the margin between +'s and −'s as large as possible. Margin= distance from separating surface to nearest point, assuming points are correct. The closest points to the separating surface are known as support vectors and they are used to help calculate the maximum margin.

Lecture 03

Rosenblatt's perceptron

Perceptron Research from the 50s and 60s

https://www.youtube.com/watch?v=cNxadbrN_aI

https://www.youtube.com/watch?v=evVSV43CRk0

training vs testing

regression

stochastic gradient

https://www.youtube.com/user/stanfordhelicopter

Transcription from Lecture

Machine Learning Tree

Perceptron Equations

$< x\ T\ x >\ w = < xyT >$

$(I + < x\ Tx>\ w) = < xyt>$

The above is a more robust error measure (Used mainly by statisticians)

Instead of E = 1/2 < ||ŷ - y||2 > we use a lower power E = 1/2 < ||ŷ - y||1 >

Another factor that would help in making the system robust would be the removal of outliers from the dataset.

Non-Linear Functions

We'll be moving on from using linear classifiers (Perceptrons) for machine learning to using streaming, non-linear functions. A use case for using these non-linear functions would be in the processing video clips. One training case for a 5 second clip would be ~20mb! Which would include around a million different vectors for machine learning.

The < x T x > matrix space is in the order of O(n2)

Stochastic Gradient Descent

$wE = 10\ 0$

As is, it's great for finding the minimum for flat curves in the weight-space. However, it's more difficult to solve for more complicated shapes.

[Topographic_1][Topographic_2]

Finding the optimum for a more complicated shape.

We undergo a process of iteration in order to find the local minima.

w(t+1) = w(t) -   wE

where   > 0

The function above is known as a naïve gradient descent as we only select one value for . A better implementation would have more values for   as we descend. > E = 1/2 < ||ŷ - y||2 >

 wE = < 1/2 w ||ŷ - y||2 > <- ŷ = f(x; w)

= < 1/2 w (ŷ - y)T(ŷ - y) >

= < 1/2 w (ŷ T ŷ) -2yT ŷ + yTy >

dE/dwk = < 1/2 (d/dwk(ŷ T ŷ) -2yT (d/dwk ŷ)) >

Dimensionality of the ŷ matrix is respective to w.

Average error of whole dataset:

E = (i)>i <- Also known as the per sample error average

Gradient descent using this sample error average:

 wE = < wÊ> > Stochastic gradient descent ***

w(t+1) = w(t) - < wÊ>

If (t) ~= 1/t, the algorithm is guaranteed to converge to a local minimum. However, we must constrain (t) like so:

O(1/t2) < (t)   O(1/t)

The SGD algorithm will yield increasingly better results after a ‘warm up’ period.

We add what we’ve gathered to a diagram of a perceptron.

[Diagram here]

s( ) = 1/1+e-

1) This equation can be considered as a good approximation for current spiking in a biological neuron
2) The following evaluation deals with the output from the perceptron diagram above: > Ê = 1/2(ŷ - y)2

dÊ/dwk = 1/2 . 2 (ŷ - y) . s’(w.x)xk

wk(t+1) = wk(t) - (ŷ - y)s’(w . x))xk

wk(t+1) = wk(t) - (ŷ - y) s’ (w . x) xk

 is considered as our learning rate. The smaller the value for   the more stable the system is.

(s’( ) = s( )(1-s( )) <— Epidemic equation)

Semi linear systems

The error function defined below is called the Cross-Entropy function. This matches well with s( ).

E = -(y log ŷ + (1-y)log(1-ŷ))

Minimum $\hat{y} = y$ If $y = 0$ as $\hat{y}$ tends 1 then E tends to infinity

In semi linear systems we can define our output $\hat{y}$ as being a probability of truth

Logistic Regression

Logistic regression can be defined as the combination of the Cross-Entropy error + Regression

Lecture 04

Calculating the gradient

Numerical differentiation dE/dw = (E(w + hei) - E(w)) / h

f'(x) = lim{h->0} (f(x+h) - f(x)) / h

programming languages don't usually have a lim construct so let h be a small number

f'(x) = (f(x+h) - f(x)) / h

where h = 1/10^9

Efficiency - gradient has m components, so you would have to calculate the above m times.

Time complexity - O(cal. E).O(cal. m) - It is not efficient

Working with floating point numbers:

1. do not add very small and large numbers.
2. do not subtract numbers of similar sizes.

The above method violates both these rules.

Back Propagation

Space: Takes up a lot of space.

Operation Count: For each primitive function the derivative must be calculated. At worst this will be a small constant vector more work.

Exact: It is exact up to floating point issues.

See Images/lecture-04 for descriptions of back propagation with unary and binary functions and functions which have one input and two outputs.

Disjoint Networks

Have a series of networks computing the same problem. If they are all making the same predictions, the probability they are getting the correct result is higher than if they are making difference predictions.

Lecture 06

6/10/2015 Backpropagation Networks in the Wild —

Backpropagation Success Stories

- PapNet http://www.medscape.com/viewarticle/718182_5 http://www.lightparty.com/Health/Papnet.html
- ALVINN http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf http://virtuallab.kar.elf.stuba.sk/robowiki/images/e/e8/Lecture_ALVINN.pdf
- RNN Handwriting Generator https://www.cs.toronto.edu/~graves/handwriting.cgi

PapNet

- Developed in 1980s
- Computer Vision techniques used to centre cells
- Features were extrapolated manually (colour etc.)
  A level of suspicion score is returned from the system
- Each returned slide would be separated with score attached

Last trick: 120 most significant cellular images (Sorted by level of suspicion) presented to human operator

Usual flaws:
* false positives * cell not fully stained * antibodies attached to it etc.

Humans could decipher these flaws easily

PapNet Success

- Much cheaper and accurate than traditional methods
- First billion-dollar company using Machine Learning

Aside

- Congress created a law to decrease false-negative Pap smear rates

ALVINN (Autonomous Land Vehicle In a Neural Network)

Push by US military to create autonomous vehicles
Biggest concern: convoys being attacked

Idea: First truck driven by person then followed by bots

First Trials: take images and try to describe attributes of the image such as *roads* etc.

Big courses were created out in desert for those autonomous vehicles

Back-Propagation Method

- Dean Pomerleau (while PhD student at CMU) Slapped a camera and laser range on the vehicles
- Early days of back-propagation

Pomerleau created a network:

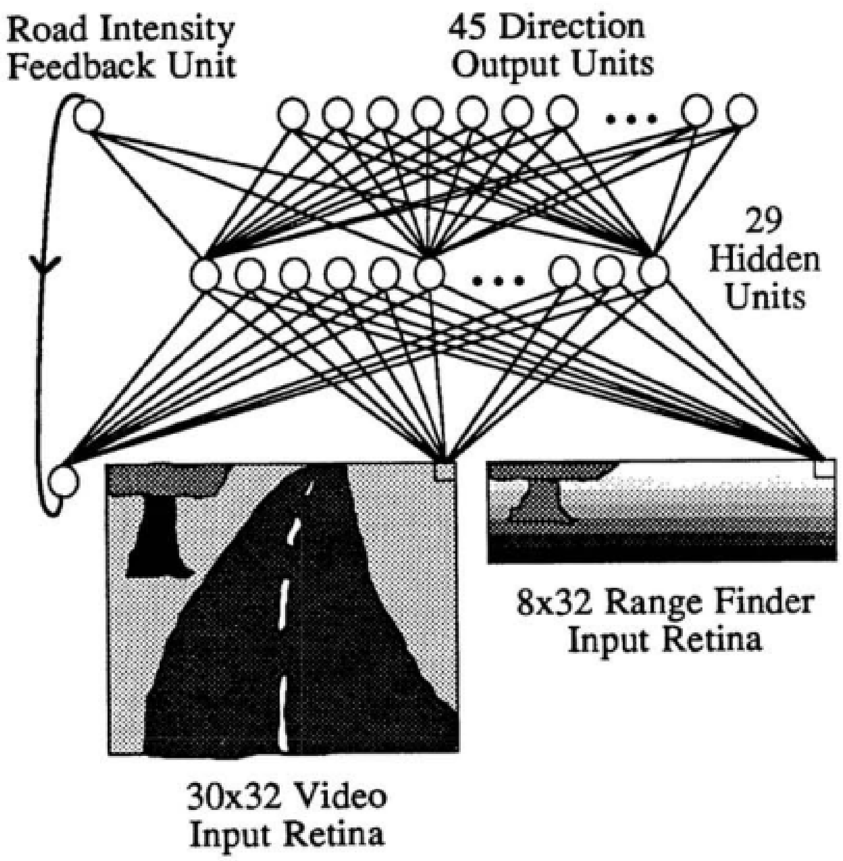

Road Intensity Feedback Unit
45 Direction Output Units
29 Hidden Units
8x32 Range Finder Input Retina
30x32 Video Input Retina

Figure 1: ALVINN Architecture

* 8 horizontal scans of 30*32 images The Range Finder returns distance of vehicle to the objects

- System returns what the steering direction should be
  Trained using Gaussian bump

ALVINN: Shallow architecture (only one layer of hidden units)

Problem: Road intensity feedback (Lightness of road compared to background)

Aside

Similar system used in Mars Rover project?
Tom Mathis worked on Mars Rover project also ###ALVINN * **One trick:** Get training data right
-Drive vehicle down road correctly once

Problems
* Unusual configuration: getting out of parking spot, parallel parking etc.

(Noise on range finder)
(Off angle - not in training set)

Interpolation vs. Extrapolation

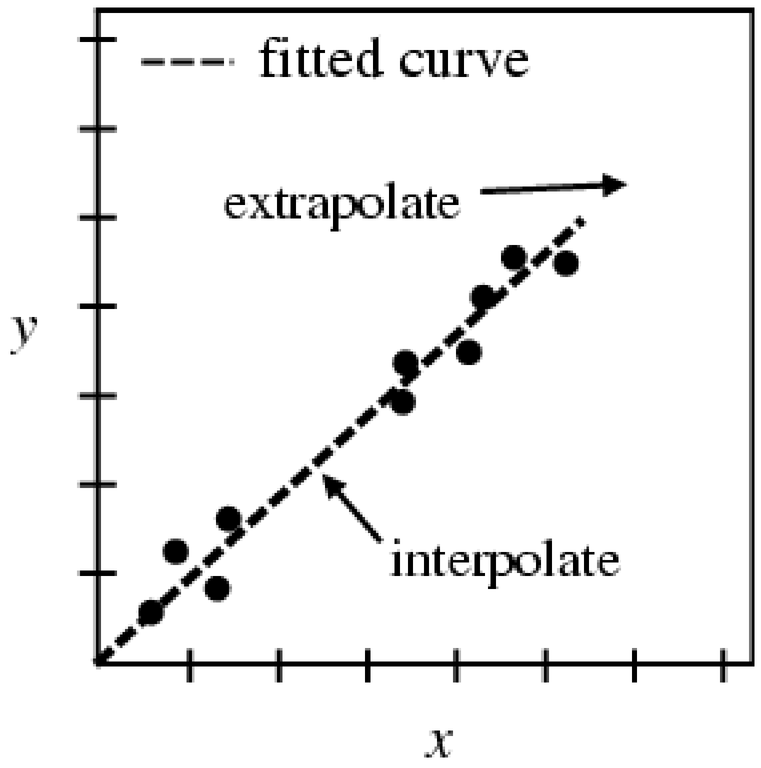


Figure 4: Interpolation/Extrapolation

- Reason why time-series is hard to create

Pomerleau: made **fake data** - moved road slightly to create new scenarios and train it on those

- Weights can correlate with the colour display from the images
- Inhibited by bright parts on some of the images
  Excited by others in different areas
- Looks for edges based on contrast of colour in the image
  Each output is weak evidence (suggestion)
  "probably a road around this way - evidence for sharp right - evidence against soft left etc."
- Tricky job for person: take all this weighted evidence and making sense of it
- Vehicle trained to drive on single pathed lane initially
  -Trip from San Fran to Seattle made by the autonomous vehicle
  (Human exerted control over steering at times needed)

ALVINN Influence

- **Google cars:** based on this model
  -new data available like GPS
- **High-end cars:** tiny camera
  System: uses above technology to notice if there is imminent crash and then takes control of engine (sees pedestrian etc.) -makes a noise to alert driver
- Single Vehicle Roadway Departure Accidents -methodology to decrease these accidents

Conclusion

- Network Weights Evolve improves after training

One criticism of this style of learning: * Similar to Black box: hard to figure out what's going on internally
(Will this system drive appropriately during an eclipse?)

RNN Handwriting Generator

[RNN Handwriting Generator Architecture] (https://raw.githubusercontent.com/GooseyGooLay/Machine-Learing-Images/master/RNN%20Handwriting%20Generator%20Architecture.PNG)

Recurrent Neural Network (useful for time-series)

- Output writes with particular style which had been fed in as input
  -elements of randomness involved
- Network trained on many styles of handwriting
- Style parameter separated at beginning
- Doesn't handle elements not in data set very well

Training vs. testing (sets)

[Training vs. Testing] (http://i.stack.imgur.com/I7LiT.png)

- Error should go monotonically downwards
  Only Care about cases which are explicit (minimise one weight's significance and put emphasis on another)
- Validation set should start out with standard error

Neural Network Halting Problem

- Minimise validation on your training sets
- Optimising rules for small sample set but over time other weights will lose significance when they should
- (**anecdote:** shared interests of people who you are to make happy changes as you suit the needs of a particular sample)
- Particular variables in training set which don't exist overall
  -leaves us with error

Lecture 07

Lecture 7 Dynamics of Gradient Descent

An important part of Machine Learning is how much time it takes

Gradient Descent's connection to Physics

The Error function in machine learning is the analogue to Energy in Physics.

Gradient descent in physical terms is like a table wobbling or a pendulum swinging. When the table is deformed there is a restoring force pushing back. Or it is like a pendulum when it is pushed there is a restoring force proportional to the distance it was moved from the point of minimum energy. It is not an abrupt transition like letting a table falling.

1 Dimensional situation

We have for a pendulum: F(x) = - c*x, where F is the force on the pendulum, x is the distance for minimum energy position.

E(x) = x0 F(s) ds = -c x0 s ds = -c s2 / 2 |x0 =1/2 c x2

In a discrete system we have the following formula for gradient descent: w(t+1) = w(t) -   wE

In a continuous physical system, we have: w(t+h) = w(t) -   h wE

Rearranging we have:

(w(t+h) - w(t))/h = -   wE

Taking the limit as h $\rightarrow$ 0 we have that: dw/dt = -   wE.

This formula corresponds to a particle going downhill with no momentum (the particle would be said to be in a highly viscous medium)

The steps taken in finding the gradient above are very small, which from a computational point of view is very bad. What we want to do is to take as large of steps as possible.

How big of a step can we take?

For a 1 Dimension system: w(t+1) = w(t) -   wE = w(t) -   d/dw(E) = w(t) -   d/dw(1/2 c w(t)2), using formula for E above = w(t) -   c w(t) = w(t)(1 -   c)

c and   must both be greater than zero. You can't choose c but can choose   (the learning rate)

We can write the above another way:

w(t) = (1 - c)t w(0)

The limitation on :

we must have |1- c |<1 otherwise it would grow and not would not have convergence of the gradient. From |1- c |<1 we get that -1< 1 - c   -2 < c   < 2/c, which is called the limit of conversion.

letting   = 1/c will give us the fastest possible convergence of the gradient.

2 Dimensional Decoupled situation

For this situation we have the following Energy formula:

E= 1/2 c1 w12+1/2 c2 w22.

Since 1/2 c1 w12 and 1/2 c2 w22 are independent of one another, the system is said to be decoupled.

We will assume only 1 value of   can be picked.

w= (w1, w2)

 wE= (d/dw1 (E), d/dw2(E)) = (c1 w1, c2 w2)

Looking at the formula w(t+1) = w(t) -   wE, we get the independent equations:

w1 (t+1) = w1(t) - c1 w1(t)

w2 (t+1) = w2(t) - c2 w2(t)

and the constraints that   < 2/c1 and   < 2/c2   < 2/max(c1, c2).

We can rewrite the formulas above as: w1 (t) = (1- c1)t w1(0) and

w2 (t) = (1- c1)t w2(0)

Let c1 < c2 then   < 2/c2

Set = 1/c2

w2 (t) = 0 and

w1 (t) = (1- c1/c2)t w1(0)

Since 1-c1/c2 will be close to 1, then you have to raise t to a very high power to find the optimum gradient. That ratio max(c1, c2)/max(c1, c2) is called the convergence limit and it determines the speed of learning.

However, a decoupled situation is not very realistic and most situations are non-quadratic. However, near the optimum it looks quadratic.

2D Coupled situation

In this scenario we will use Taylor series around the point w*, the point at which the maximum gradient is.

E(w*+ Δw) = E(w *) + 0 + 1/2 ΔwT 2wE(Δw) + O(Δw 3).

we get 0 as the second term since wE = 0 at w*

Set H = 2wE

For the previous decoupled situation, we have:

H = [[2E/ w12, 2E/ w1 w2], [2E/ w22, 2E/ w2 w2]] = [[c1,0], [0, c2]], that is, the 2×2 diagonal matrix with c1 and c2 along the main diagonal.

Hence 1/2 ΔwT H Δw = 1/2 Σi Σj 2E/ wi wj Δwi Δwj.

Diagonal Matrices are easy to work with. We can choose a new coordinate system (a new basis) such that we always get a diagonal matrix through the use of eigenvectors.

 E(w* + Δw) = H Δw + O(Δw 2)   E(w* + Δw)   H Δw

Let vi be an eigenvector of H then Hvi = i vi and we can express Δw in the eigenbasis:

Δw = Σi bi vi

Then H Δw = H Σi bi vi = Σi bi H vi = Σi bi i vi.

Hence the formula for bi(t+1) = i -   i bi(t) with   < 2/(maxi i)

The “condition number” of H is max / min.

Lecture 08

Lecture 8 Notes

When we are close enough to the optimum the formula becomes quadratic and convergence rate depends on the ratio max / min

Grading descent equation from previous lectures:

w(t+1) = w(t) -   E (where   is learning rate)

Depending on our chosen learning rate we can have different outcomes:

a) learning rate is too big and convergence will never happen - with every step of descent we will move further and further from the optimum:

b) learning rate is too small, convergence will happen but this will be a slow progress and will take a lot of iterations of the algorithm

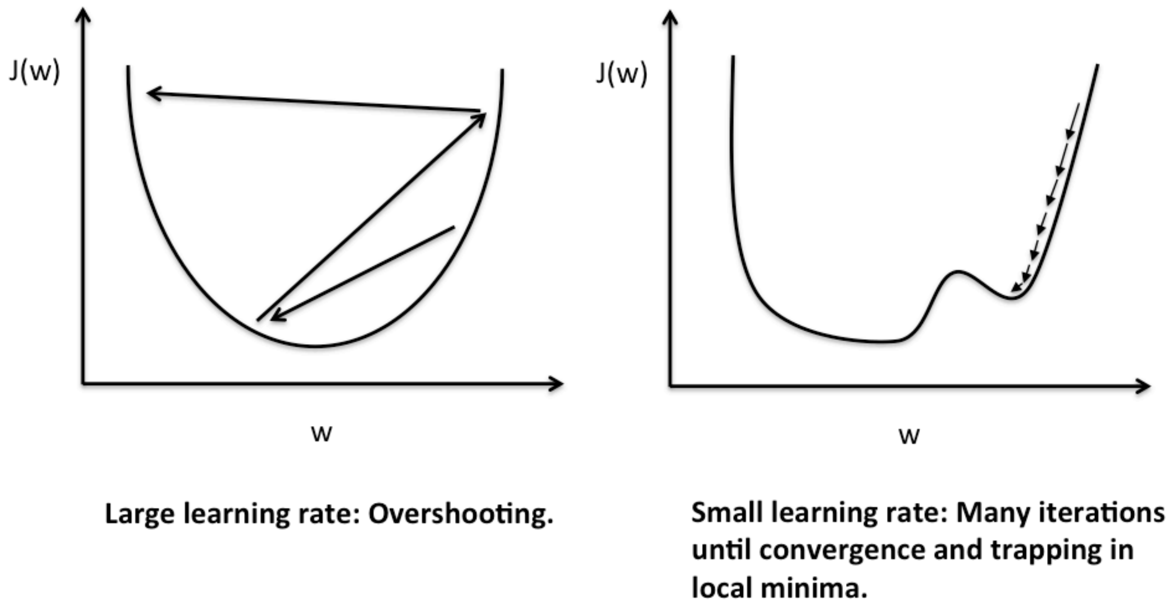

J(w)
w
Large learning rate: Overshooting.
J(w)
w
Small learning rate: Many iterations until convergence and trapping in local minima.

Figure 5: Choice of learning rate

c) well-chosen convergence rate will converge and do so reasonably quickly

To obtain such optimal learning rate we will introduce new variable - , for momentum, such that

0   < 1

(at zero we have no momentum at all, and at 1 momentum will not stop at the minimum, we need some friction to slow it down once we've reached our goal)

If we add this new factor our formula becomes:

w(t+1) = w(t) -   E +   (w(t) - w(t-1))

Simplify:

w(t+1) - w(t) = -   E + (w(t) - w(t-1))

Rewrite w(t+1) - w(t) as Δw(t) and substitute it into the formula above:

Δw(t) = - E + Δw(t-1)

where for stability < 2 / max

There has to be a balance in setting the momentum to apply optimally to both max and min. If max is a lot bigger than min we calculate momentum using the following formula:

Δbmin = - E/ bmin + Δbmin

Δbmin - Δbmin = - E/ bmin

(1 -)Δbmin = - E/ bmin

Δbmin = - /(1 -) E/ bmin

if is too high it would affect our max which will have it overshoot the minimum.

However, in practice, for big data sets there are too many calculations to use batch grading descent. Instead, another option is to use stochastic grading descent:

Δw(t) = - Ê

(where Ê is E plus some bounded amount of zero-mean noise)

- has to go to zero, slowly enough that we can get rid of the noise.

Σt=1...∞ (t) = ∞

(to have enough momentum), but also, for descent to be fast enough

Σt=1...∞ (t)2 < ∞:

so

O(1/√t) < (t) O(1/t)

In practice stochastic descent is not used either. We can't always get an optimum, for an algorithm that uses real life changing data a good approximation is what we aim for.

All gradient descent methods are weak, it is much better to analytically find the optimum and just "jump" there.

Support Vector Machines (SVMs):

MLP require a lot of tuning. They are hard to implement and there's a lot of decisions that we have to make. New approach - SVM (developed by Vladimir Vapnik and his colleagues) Some of the SVM logic can be added to MLPs to enhance their performance. Data example:

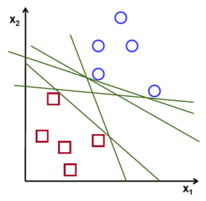


Figure 6: data example for fitting linear classifier

For data like this we can fit many possible linear thresholds that will predict different values. If we had to choose one, we could find a place, such that: 1) classifies data correctly 2) positioned in such a way that the nearest square and the nearest circle are as far away from it as possible.

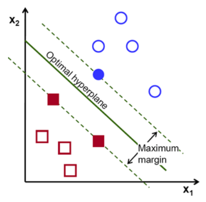


Figure 7: data example for fitting SV

Its position depends solely on data points touching the margin (the support vectors).

SVM - linear classifier with maximum margin. (The bigger the margin, the better)

Lecture 10

20/10/2015

Support Vector Machine (SVM)

Last Lecture: Maximum margin problems * Introduction of soft margin
When there is mislabelled data; a hyperplane is introduced to cleanly split data and maximise margin distance

- Problem based on linear classification
  Margin is straightforward to calculate in linear case but not when the problem's non-linear:

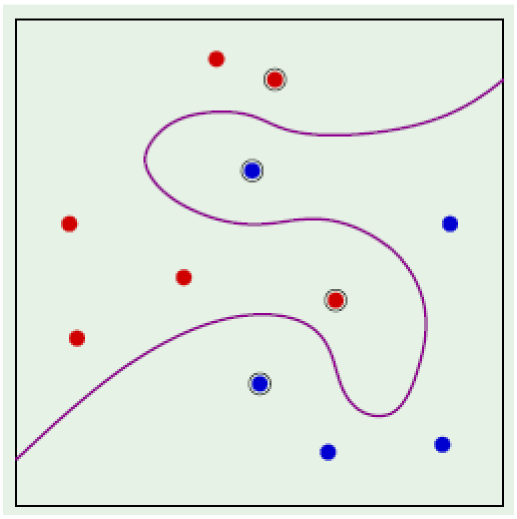


Figure 8: Non-linear margin

Kernel Trick

- Map observations to a higher dimensional space using a Kernel Function:
  $k(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{V}}.$
  e.g.

- (x) - Intractable (heard to calculate by itself)
  Analogy - GPU input vector which you cannot alter once it is being processed

Kernelize the Algorithm

- Instead of operating in input space - change x's to (xi)'s to move to feature space

: primal/single representation of the vector
: dual representation of the vector
i: slack parameter
, , - penetration variables (penetrate margins)
y(i) = ±1 for either "yes"/"no" class of data points

Math Breakdown:

Kernelized Algorithm

- Dual representation ( ) used for quadratic programming problems as opposed to primal representation ( )

Kernel Function

Mercer's Theorem

Pre-condition:
If k is symmetric:

,non-negative definite:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} K(x_i, x_j)c_i c_j \geq 0$$

for all finite sequences of points x1,..., xn of [a, b] and all choices of real numbers c1,..., cn
Post-condition:

Examples:
Identity Kernel:

- takes O(n) work in n-space

Polynomial Kernel

For degree-d polynomials, the polynomial kernel is defined as:

$$K(x, y) = (x^{\mathsf{T}} y + c)^d$$

where x and y are vectors in the input space and c   0 is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial.

- When we used quadratic kernel we dropped all linear terms
- No arguments about which kernel function to use as you can always use them both (add them up) Example:

Gaussian Process

- Gaussian Kernel:

- Can be used to compute similarities between images
- Fee for using this: maps to infinite dimensions

Kernel Function applications

- Find similarities between two pieces of text

*When we finished the optimisation above:
Problem: there were no x's left, just i's and j's

When we want to embed SVM in your system (sneeze function in camera):

We only need to store the support vectors of people sneezing from the training set
Only download these into the camera:
e.g. 200/1000 training cases

SVM Conclusions

Popular kernels: quite robust
-Reasons why people like SVM instead

Positives: * Beautiful Math (Kercher's...) * SVM depends on number of support vectors
- can work in higher-dimensional space as only looks at subset of vectors * Turn Key

Criticism: * Glorified template matching

Lecture 11

2/11/2015

Lecture 11 Notes

Unsupervised Learning ##### Clustering

Linear vector quantization (LVQ) aka k-means

Example. You are given a task of calculating an average weight of a mouse in some mouse colony. How would you approach it? You could measure the weight of every mouse, sum it up and then divide the total by a number of mice measured.

If there are too many mice to measure, you could, perhaps, measure some representative sample of the mouse population and calculate the average based on the sample.

But what would you do if the average was dynamic, for instance, the mice are a subject of the experiment that affects their weight and we want to track the changes to the average weight? Keep in mind, your solution needs to be realistic in terms of data storage we can designate for our calculations.

To calculate a running average, we simply need to keep track of the running total and the current count:

> a1, …, at are weights
>
> St = Σi=1…t ai is a total weight

and then the average weight is calculated as follows:

> At = St/t

if we need to add another mouse's weight the next day we update out total and our count:

> St+1 = St + at+1
>
> At+1 = St+1/t+1

In our case, however, if we want to track the changes in the average weight, we may want to only calculate the average weight over the last month.

We want to disregard old data and give more weight to the new data. To do so we can use decaying sum:

Decaying estimate:

> St = at + 0.98 at-1 + (0.98)2 at-2 + … = Σj=0…∞ (0.98)j at-j

We call (0.98) in the following example decaying constant and we can denote it as , s.t 0 < < 1

We adjust our formulas accordingly:

> St+1 = at+1 + 0.98 St
>
> At = St/Σj=0…∞ (0.98)j

Now let's say in our example we have 2 subpopulations of mice and they differ by weight. In this situation we would like 2 averages, each calculated for a separate subpopulation.

How can we begin to sort them?

Ideally, we would like to be able to classify a mouse as belonging to one or the other subpopulation and only update the relevant subpopulation average with that weight. But we don't know, when we are given a mouse, which class of mice it belongs to. One algorithm gives us a way around it by creating cluster centres and using the cluster centres for classification.

k-means Algorithm

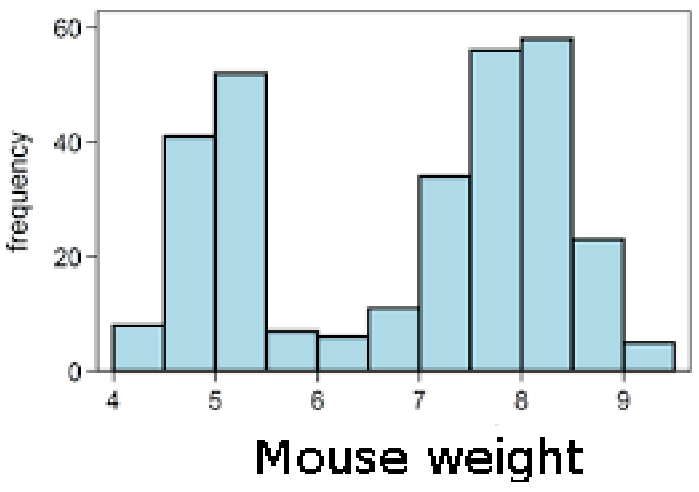


Figure 9: bimodal histogram

Using this algorithm, we associate the value of every new sample with the value of the cluster centre closest to it.

```
loop: > read in xt
        calculate ĵ = argminj ‖ xt - wj ‖
        wĵ ←   wĵ + (1 -  ) xt
```

And as we add this new value to some cluster, we update the average value of that cluster and move wj closer to the added value.

As the algorithm progresses, some clusters will starve, some will crawl towards each other, they will not be quite as they were during the initialisation, but resulting clusters will be fairly accurate.

If we initialise many more clusters than we expect we need, and if they are quite spread out, we can arrive to the correct positioning of cluster centres by throwing away starved ones and joining the ones that are close together.

Usages of k-means:

One example of using this algorithm could be recognition of hand written digits.

Another one - speech recognition.

When the system analyses speech for each bin it gets a vector (of ~ 128 dimensions) to represent frequencies. It is hard to compute things using 128 dimensions, so instead of this vector analysis works on cluster indices generated for the vectors. When a new user starts using the system, the clusters get updated to suit.

In clean form, however, clustering is rarely used.

It can be used for cell sorter to find abnormal features in tissue, or in biology, clustering samples into species, or purchase history. In real life clustering is combined with other algorithms to improve its accuracy. Factors to consider when using clustering algorithms:

- is clustering going to converge?
- will the pull factor lessen with time and will the centres move less?
- will the resulting clusters be correct? We can use Voronoi diagram to split our space into planes where the points lying in the same plane correspond to its cluster centre:

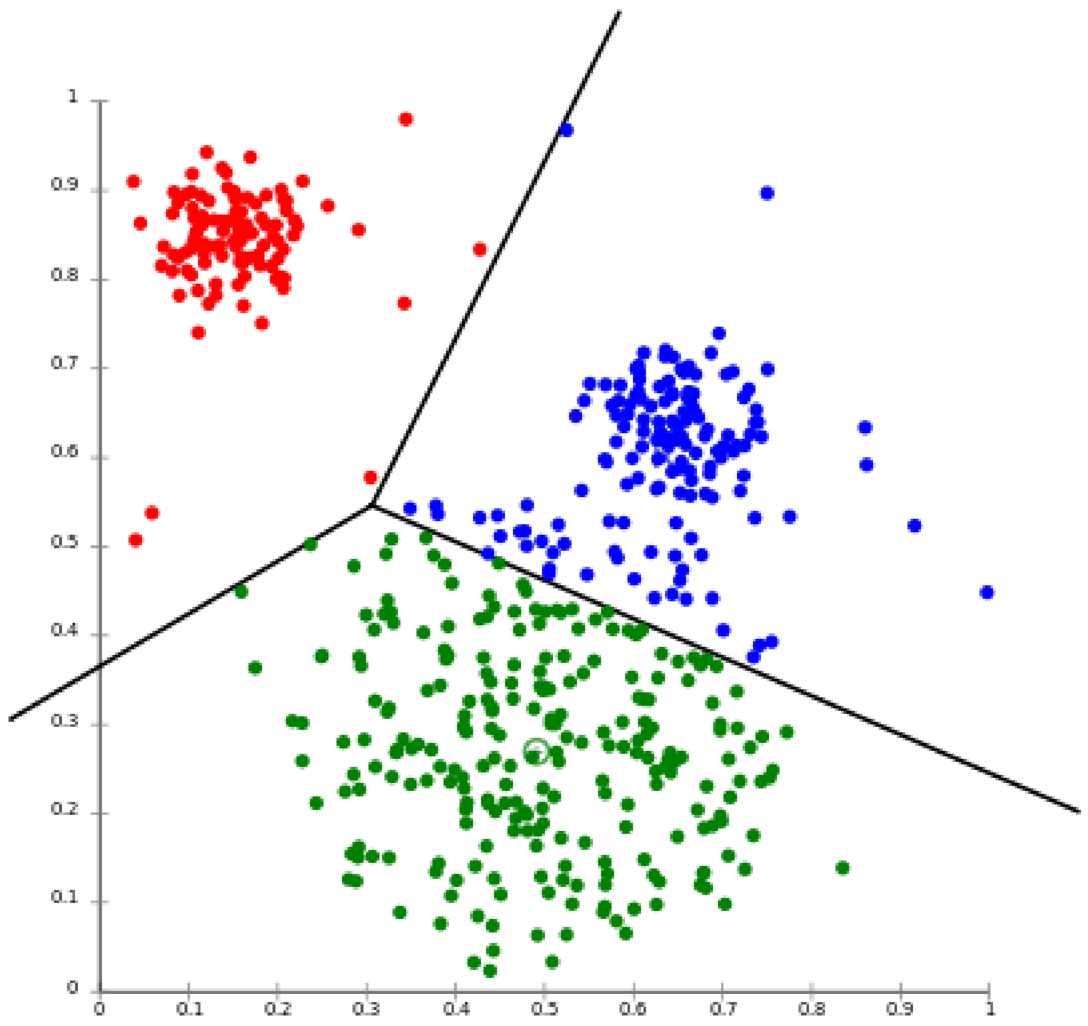


Figure 10: Voronoi diagram

To find cluster centre wj amongst many cluster centres such that x is closest to wj we minimise error E using stochastic gradient on clustering function:

$E = \Sigma t \parallel xt - w\hat{j} \parallel 2,$

where > $\hat{j} = \text{argminj} \parallel xt - wj \parallel,$

hence > $E = \Sigma t \ \min\Sigma j \parallel xt - wj \parallel 2$

Lecture 12

3/11/2015

Bayes' Rule, Coin Toss and K-Means

The proportion of tall people with cancer > Population of people with cancer within entire population

http://www.futilitycloset.com/2015/10/19/more-fun/

Q: If tall people are more likely to get cancer, then are people who get cancer more likely to be tall?

So, yes people who get cancer are more likely to be tall.

Bayes' Rule

Imagine a machine, with various parameters, that produces data randomly, and a database of some kind of documents.

When run, it produces a one-page document.

There is a very small chance that it will match a document already in our database.

However, if we set the parameters a certain way, our machine may produce a document similar to docs in the database.

How do we set these parameters properly?

Setting the Parameters on our documents machine

Discrete Distribution

Consider a coin flipping machine, with a 0-1 probability parameter. The machine can either produce 0, for heads, or 1, for tails.

It has a binary dataset (y1, ..., yn)

And the machine outputs (ŷ1, ..., ŷn)

In a dataset of ~100, the chances of ŷ = y is vanishing small.

Note that we can store vanishing small numbers as logarithm.

If the machine has memory, then every coin flip will be like taking marbles from a bag, and not replacing them

-> every previous result will affect the next result.

However, this is not how coin tosses work, every coin toss outcome is independent, so our machine has no memory.

K-means

K-means is an unsupervised learning algorithm that classifies a given data set into clusters.

We can imagine the dataset is produced by a machine that will populate a graph with data.

The machine has three inner machines, each of which will produce a data point that is around a certain area, but has some degree of randomness or noise.

Using K-means, we want to look at this data set and find out which points were created by which machine.

We will look at K-means further in Lecture 13.

Lecture 13

Mixture Model

Two iterations of:

- (Data, Assignments) —-m-step—-> Model
- (Data, Model) —-e-step—-> Assignments

Hidden Markov Model

- Finite state machines
- Baum-Welsh speech recognition for NSA
- Outputs can be associated with states or transitions
- Bernoulli trial and speech recognition examples
- = probability distribution for the start state

What makes it "hidden"?

- States are hidden - only have the output
- Goals (given output):
- Reconstruct sequence of states
- Determine which model produced it Bayes' Rule

Problems:

1. (output seq, HMM) –infer–> sequence of states
2. (output seq, HMM) –infer–> P(output seq | HMM)
3. (output seq(s)) –infer–> HMM (figure out the parameters of HMM) (EM steps)

Markov Property

- Probability of the next state is independent of previous states - only depends on the current state.

Cheat Sheet

- Tables drawn for:
- Forward algorithm (alpha)
- Backward algorithm (beta)
- http://barak.pearlmutter.net/misc/hmm.pdf

Lecture 14

Coupled Hidden Markov Model

HMM can have discrete values

No efficient algorithm is known for the Coupled HMM, there is known algorithms for single HMM

Multiscale Quadtree

-Images have multiscale properties, i.e. the statically probability of the distribution remains the same when viewing the image when zoomed in or viewing it full sized.

Stereo fuse (Stereo vision) - Using two cameras that are close side-by-side position for depth perception, gives uncertainly which is important, e.g. we know where a ball might land after thrown but can't tell for sure where it'll land but the general area, i.e. a gust of wind could change the trajectory at the last second.

Encoding on noisy channel

$m > n$

Message matrix $< m >$ | n - message ˆ Parody (XOR of all bits)

Best matrix for encoding messages on a noisy channel

Above matrix m is the only known one to approach Shannon's limit of transmission

A Graphical model allows us to use inference to get x

(x1) → (x2) → (x3) …… ( ) → ( ) ↓ ↓ ↓ ↓ ↓ (y1) (y2) (y3) ( ) ( )

These tables have P(V = Vi | Ai = ai, A2 = a2 …)

```
##v * #a2 * #a3 * ... * an
```

Coin flip table

V	a1	a2
	0	0
	0	1
	1	0
	1	1

directed graphical model -> undirected graphical model Using energy instead of probability

given ancestors of node, find node

write byte read byte

(x) -> (y)

P (y | x)

table for directed model

Energy Model

Pxy P00 = 0.45 P01 = 0.05 P10 = 0.1 P11 = 0.4

P(X, Y) = P(T|X)PX

P( ) = 1 / z * e - (E) / x

set T = 1

log P = -log(z) - E E = -log(z) - log(P) z = Σ ˆ(e-(E)/T)

P /P 1/z(e) - (E)/T = eˆ(-E +E) = -E + E = log (P)/P = log(P) - log(P) ________________________
1/z(e)-(E)/T

log (P00/P01) = -E00+E01

E01 = E00 + log (P00/P01)

Lecture 15

Intro: Previously on CS401...

Machine Learning theorists have a probabilistic perspective. Previously we have looked at probabilistic algorithms such as:

- Bayes rule
  - P(A|B) = P(B|A) P(A) / P(B)
  - relates the odds of event A1 to event A2, before and after conditioning to another event B.
- K-means clustering
  - unsupervised learning algorithm that classifies a given data set into clusters.
- Back propagation
  - calculates the gradient of a loss function with respect to all the weights in the network.

Now: Probabilistic Estimation with Confidence Interval

Augmenting the backpropagation network

Nice to know how certain our output is

Gaussian distribution of output?

[image placeholder]

[derivation placeholder]

Unsupervised Learning

- Data fit to probability distribution
- Graphical models
  - graph theory
  - inference is hard for graph models

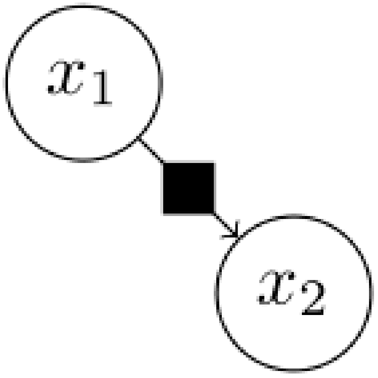
[image placeholder]

Lecture 17

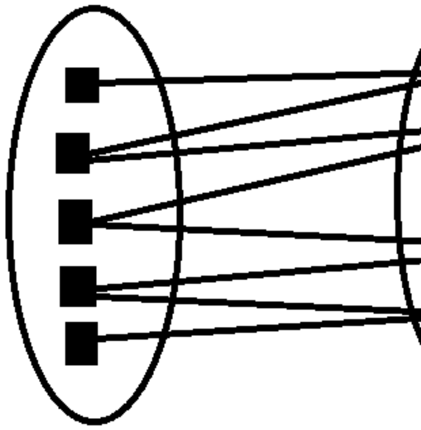
How to sample a Graphical Model

The Product-Sum algorithm is an algorithm for computing for sampling a graphical model. It works well if the graph has a tree like structure, but in other situations it isn't. It is sometimes called a message passing algorithm.

- Add in extra nodes into your graph called "factors" in the middle of each arrow. (The original nodes are called variables). Also you can drop the directness of the graph.



- This is now a bipartite graph (i.e. one which can be coloured with only 2 colours and all adjacent



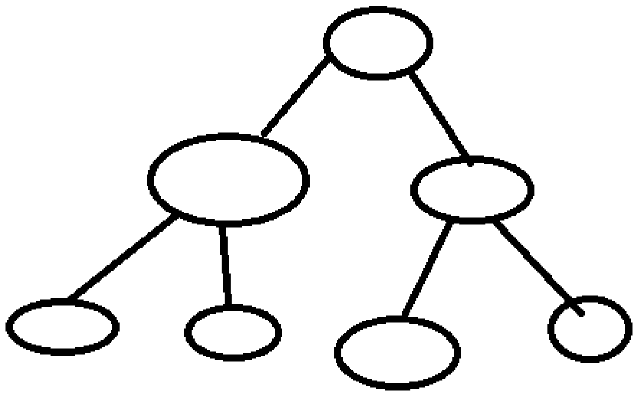
  nodes have different colours. Associate numbers with all nodes.
- Alternate updating these numbers. The Factors get updated by products of terms and the Variables get updated by sums of terms.

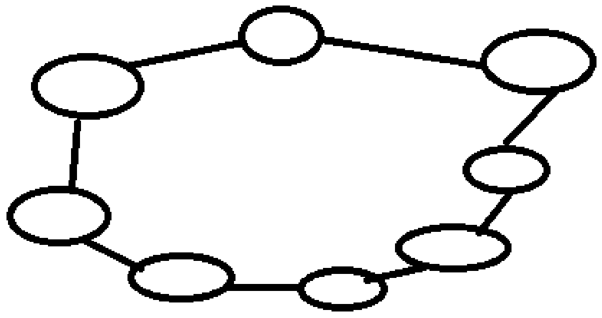
In Practice We have made some observations (e.g. characters) and we want to figure out the marginal probabilities in the graphical model (e.g. the word someone was trying to type). Finding the full joint distribution is too hard of problem for graphical models

- We associate a function f(xi, xj) with the factor between xi and xj.
- Hence we can say P(xi, ..., xn) = &##928;jfj(xa(j), xb(j)).
- Update the factors with a message called v. v is a message from a variable xn to xm and it represents the conditional probability P(xm| all the children of n)

Problems

With a tree graph, the product-sum algorithm will converge. However, with a chain of loop like structure old information will get amplified. If chains are long or there is some sort of attenuation this is not really important, but if the chains are short the probabilities you get will be wrong. Tree structure:

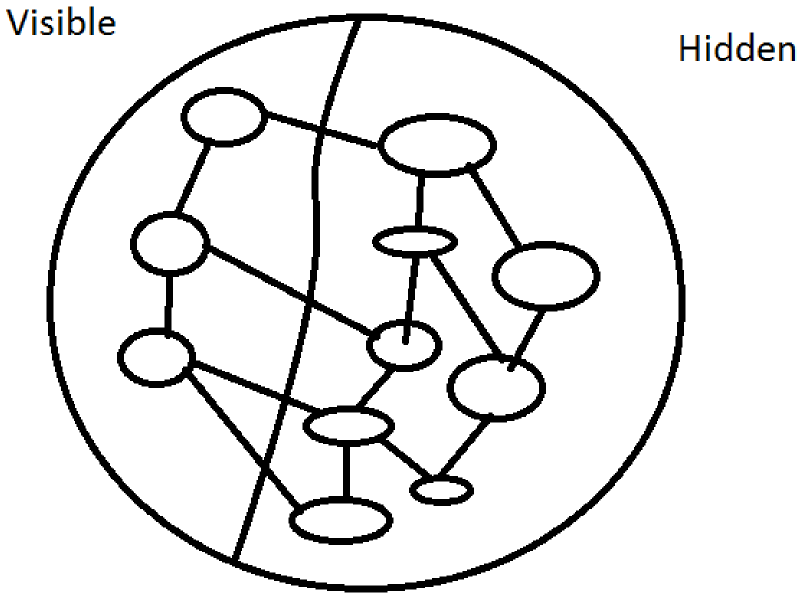




chain structure

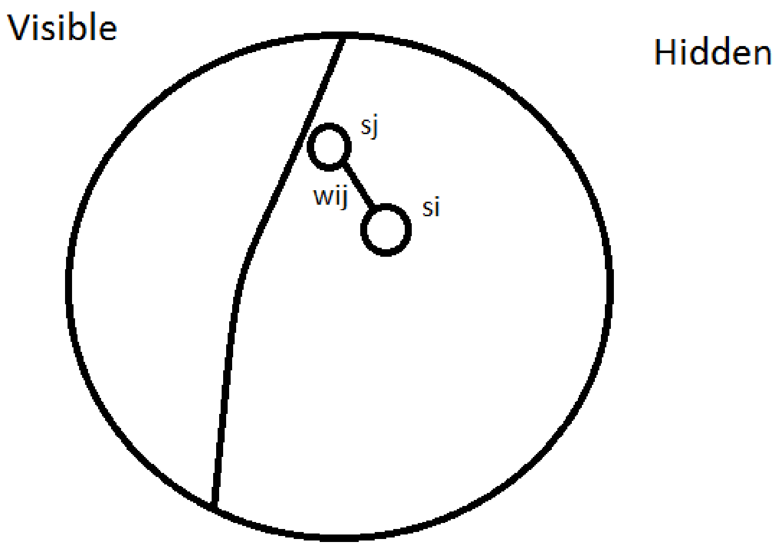
Boltzmann Machine

Problem Given the structure and lots of samples from Graphical models, can we figure out the relation-



ships between them? This is a learning problem.

From Boltzmann Machine we will deal with binary variables. The graph is separated into 2 parts, the hidden variables and visible variables. The links in the graph encode a complicated probability distribution. The hidden part is supporting structure for the visible.



$$E = \sum_{i<j} s_i s_j w_{ij}$$

Let   be some configuration of the system. P( ) is proportional to e-E /t

Pick si and consider it changing its state.

Calculate Esi = 1 and Esi = 0

$$\frac{P(\alpha[s_i{=}1])}{P(\alpha[s_i{=}1])+P(\alpha[s_i{=}0]))} = \frac{e^{\frac{E_{\alpha,s_i=1}}{T}}}{e^{\frac{-E_{\alpha,s_i=1}}{T}}+e^{\frac{-E_{\alpha,s_i=0}}{T}}}$$

$$= \frac{1}{1+e^{-\frac{(E_{\alpha,s_i=1}-E_{\alpha,s_i=0})}{T}}}$$

$$= \frac{1}{1+e^{\frac{-\Delta E_{\alpha,i}}{T}}}$$

$$\Delta E_{\alpha,i} = E_{\alpha,s_i=0} - E_{\alpha,s_i=1}$$

$$= \sum_{j,\text{ connected to i}} s_j w_{ij}$$

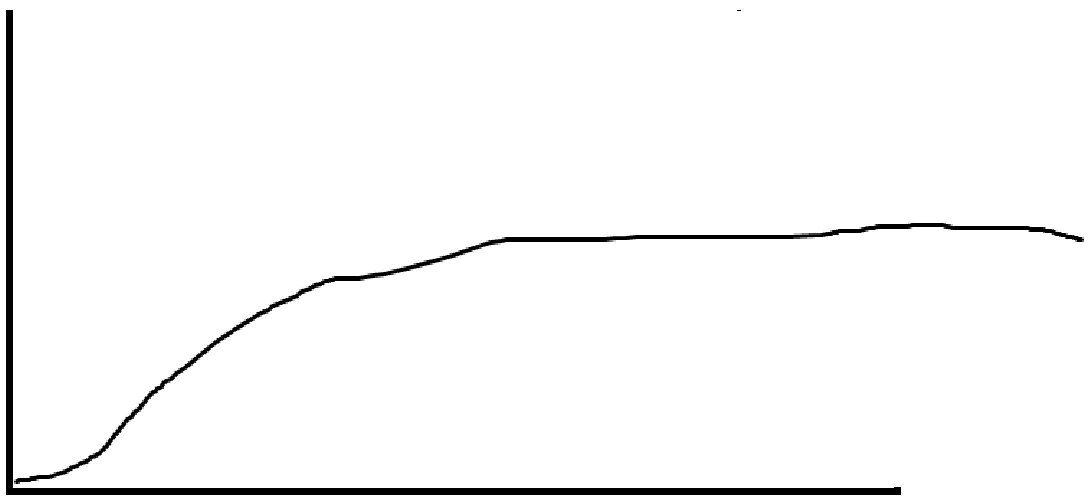


Figure 11: graph of E

The parameter T can be viewed as temperature

A High T gives:

A Low T gives

let be a configuration of the visible states and be a configuration of hidden states

We want to calculate:

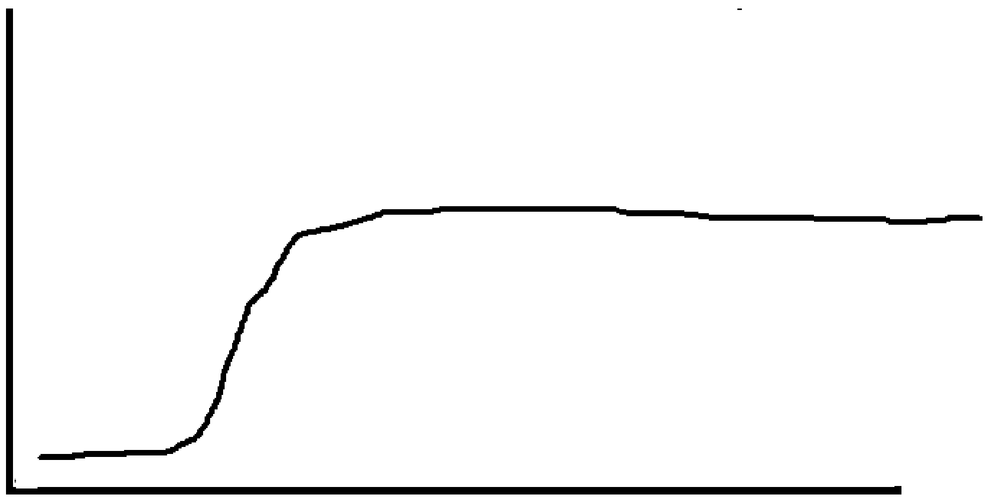


Figure 12: high temperature

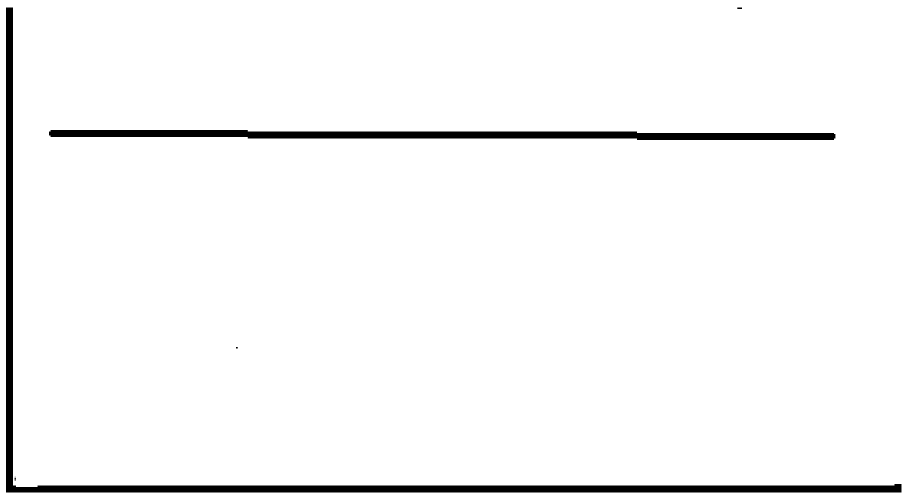


Figure 13: low temperature

$$G = KL[P^+(\alpha)|P^-(\alpha)]$$

$$= \sum_\alpha P^-(\alpha) \log(\frac{P^+(\alpha)}{P^-(\alpha)})$$

Figure 14: formula for G

$$\frac{\partial G}{\partial w_{ij}}$$

$$= \frac{1}{T}(P_{ij}^+ - P_{ij}^-)$$

$$P_{ij} = P(s_i, s_j)$$

Figure 15: formula for derivative of G

Lecture 18

Matrix decomposition

Recap

- Graphical models and Bayes nets => supposedly the future of ml.
- However inference from graphs is considered an intractable problem.

Matrix decomposition

+In general matrix decomposition is the factorization of a matrix into a product of matrices. In particular, we talk about non-negative matrix factorization(NMF), that is we factorize a matrix into two matrices such that all three matrices are element wise non-negative.

+Many problems can be rephrased as matrix decomposition and thus it is a handy tool to have in the machine learning shed.

+Data sets such as images etc. fall into the line of fire of NMF

Example

Consider the 2D data set

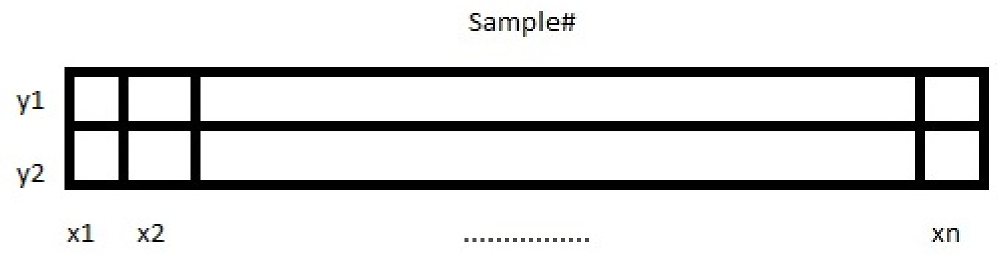


Figure 16: 2DData

where y1 and y2 are drawn from Gaussian (normal) generators as follows:

```
+y1 ~ g(0;1^2)
+y2 ~ g(0;2^2)
```

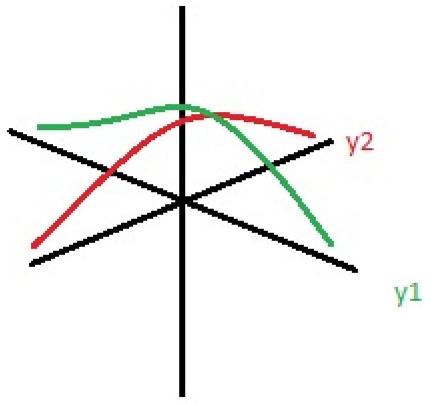
where a Gaussian distribution is parameterised as g(mean; std. dev)

we also have the following energy functions for y1 and y2:

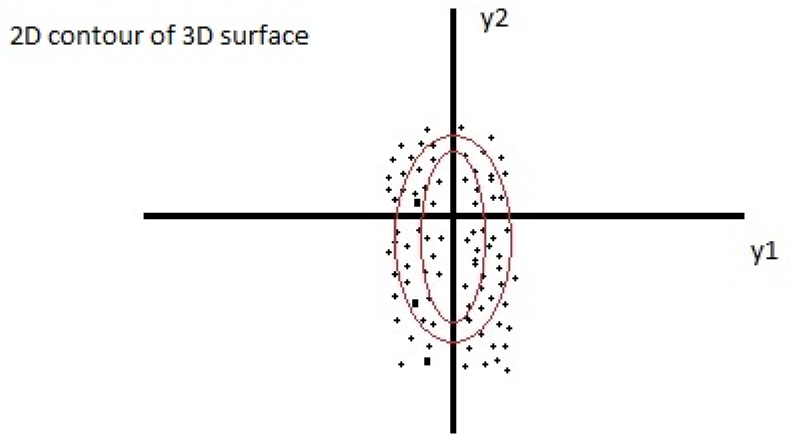
```
+E(y1) = y1^2
+E(y2) = (y2^2)/4
```

(for those who are wondering what all this talk of energy functions is about and what they have to do with stats this might help.)

We can visualise the data using a histogram:



or with a scatter plot:



The physics analogy here is the distribution of molecules in 2 different rooms. We want to consider them separately, then consider the joint distribution of molecules across the two rooms.

Thus the joint energy of the two samples is:

```
E(y1, y2) = (y1^2) + (y2^2)/4
```

Then we can get the probability of some X across the two Gaussians:

```
p(x) = 1/Z(e^-(E(y1, y2)))
where Z is the partition function (see previous lectures)
```

Non axial parallel example

In the case where the distributions are non-axial parallel. For example:

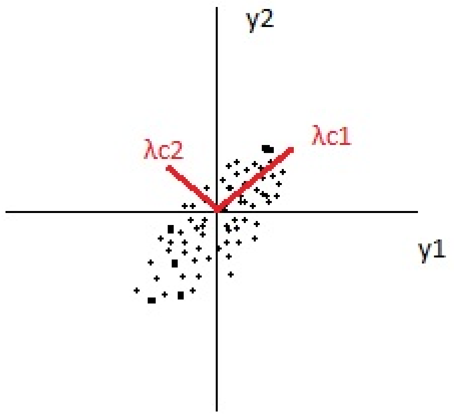


Figure 17: nonaxialp

There are 2 sources of variation in the data given by ci, where:
||ci|| = 1 //ci is a unit vector
 i = amount of std. dev. in direction i

Thus we have another energy function, that describes a 2D Gaussian with arbitrary direction:

$$E = \frac{1}{2}\left(\frac{1}{\lambda 1}(c1 \cdot x)^2 + \frac{1}{\lambda 2}(c2 \cdot x)^2\right)$$

`where:`

$c_i \cdot c_j = [i == j]$ // [i=j] Knuth notation, returns 0 or 1

Expanding to n-dimensional Gaussians

How do we expand this concept to n-space?

$$X^T (c1|c2|....|cn) X => \sum_{n}^{i=0} (ci \cdot x)^2$$

Figure 18: ndimengaussianorientation

Include the lambda's:

$$E = \frac{1}{2} X^T C^{-1} X$$

where

$$C = (\lambda 1 C1|....|\lambda n Cn)$$

When the distribution(s?) are axial parallel we have:

$$\begin{bmatrix} \sigma^2 & \ldots & 0 \\ 0.. & \sigma^2 & ..0 \\ 0.. & .. & \sigma^2 \end{bmatrix}$$

Figure 19: axialpmatrix

when the distribution(s?) are not axial parallel the matrix gives you the orientation of the distribution(s?)

Going back to our 2D data set, we have x expressed as a sum:

$x = (x1 \cdot c1) \cdot c1 + \ldots.. + (xn \cdot cn) \cdot cn$

Example

---

say we have an 8 * 8 matrix X, a 64 * 12 matrix A (the column joined ci's) and a 12*M matrix B
we want to decompose X (approximately) S.T:

$X \approx A \cdot B$

we find:

$argmin_{AB} \, ||X - AB||_F^2$

i.e. the least squares approach
with the constraint that:

$A^T A = I$

(aside: all major contributions to the field of statistics have been by psychologists, not statisticians.)

What if we relax the above constraint (i.e. that the ci's are pairwise orthogonal)?
=> No unique solution (to what???)

For example, consider the distributions:

Usually only one source of variation is non-zero in this case. This allows for independent component analysis.

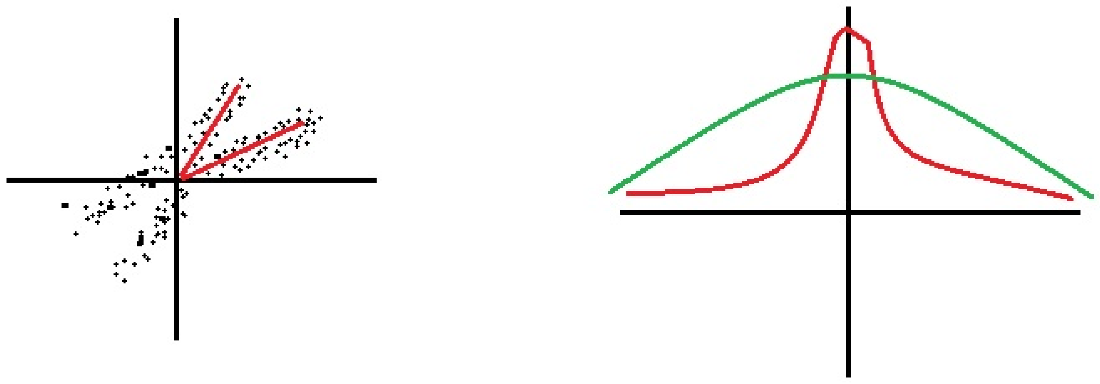


Figure 20: noorthconstraint

Lecture 19

More Matrix Decompositions

PCA and SVD

Note readable intro to PCA: http://efavdb.com/principal-component-analysis/

Independent Components Analysis (ICA)

https://en.wikipedia.org/wiki/Independent_component_analysis

http://www.inf.fu-berlin.de/lehre/WS05/Mustererkennung/infomax/infomax.pdf

Nonnegative Matrix Factorization (NMF)

https://en.wikipedia.org/wiki/Non-negative_matrix_factorization

http://www.nature.com/nature/journal/v401/n6755/full/401788a0.html

http://www.columbia.edu/~jwp2128/Teaching/W4721/papers/nmf_nature.pdf

Lecture 20

Independent Component Analysis

Wikipedia Article on ICA

Example:

We have a group of people betting on a game. Our objective is to minimise the loss we make while betting.

- There are multiple rounds of betting.
- You must bet on every game.
- One person in that group is an expert and is always correct.

Solution: Find the expert as quickly as possible.

xi(t) = prediction of the ith person in game t wi(t) = (boolean value) whether we listen to the ith person's advice z(t) = the winner.

Idea: Bet on the side which most people bet on. Only consider the bets of those people who have always been correct up to this point. > y(t) = [Σi (wi(t) xi(t)) > 1/2 Σi wi(t)]

Update the weight associated with each person (only those who have always been right before this game). If they guessed incorrectly in this game set their weight to 0 else 1. > wi(t+1) = wi(t) [xi(t) = z(t)]

Regret: How much we will lose before we find the expert. Regret <= log2n

Update Problem: No one is always correct but some people are better than others at guessing.

wi(t) = weight associated with the ith person's advice.

Idea: Reduce the weight an individual has by 1/2 if they bet incorrectly. Keep their weight the same if they guess correctly.

    wi(t+1) = wi(t) (1/2 [xi(t) = z(t)] + 1/2)

Lecture 21

Sparse Learning and Boosting

Winnow

Winnow Algorithm

Strong Learnability

Probably Approximately Correct (PAC)

Probably approximately correct learning

Almost certainly generalizes really well

Weak Learning

Has a slight chance of generalizing slightly better than chance.

Boosting

Boosting

AdaBoost

AdaBoost

Lecture 22

Stepping on the Curve

ROC Curves

ROC Wiki, ROC Curve

- Became popular during WWII at the invention of radar / sonar.
- Gives signal feedback (approximation as intensity of return + check if it exceeds some threshold).
- The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR).

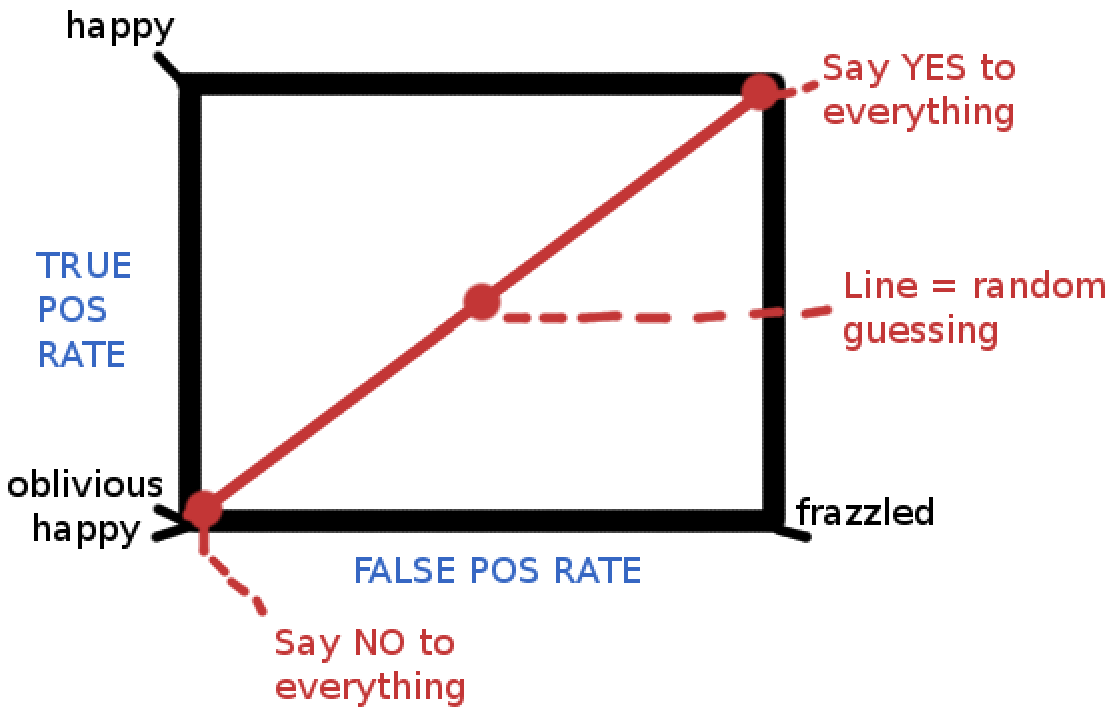
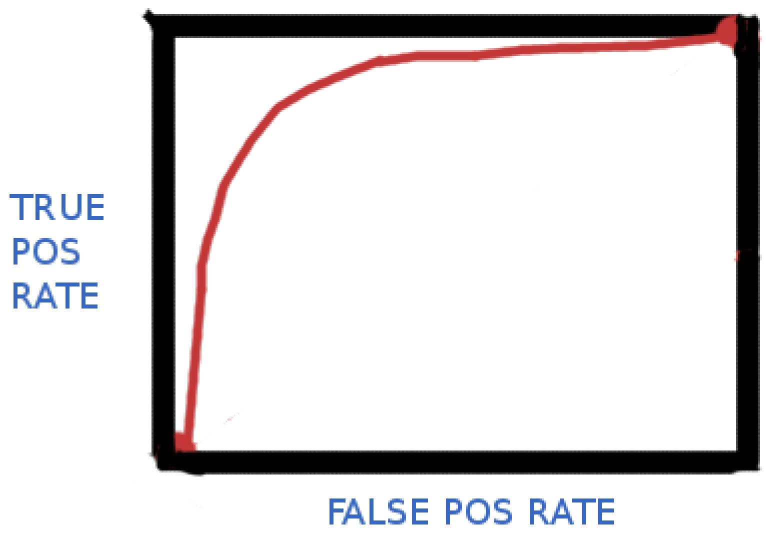
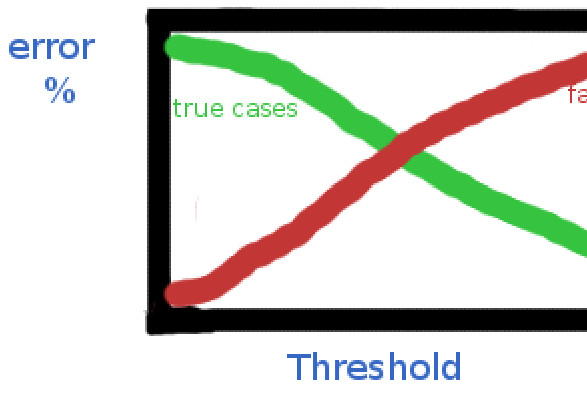


Figure 21: ROC

- The above is an idea of how these graphs will work. We generally pick samples from above the line. We will never have sample that look this straight. In general, we will end up with a curve like this, where the first image comes from taking a slice of the second image:





- Consider why these graphs might be used. If we're testing for cancer, we want to ensure there are as close to 0 false negatives as possible (as this would mean an incorrect diagnosis) so we could choose a point around the top left.
- We might also be dealing in stocks, in this case, we want to never lose money, but don't care how little money we make at a time, so in this instance we can allow for some false positives.
- Finding the balance for what you need is key.

An actual ROC curve:

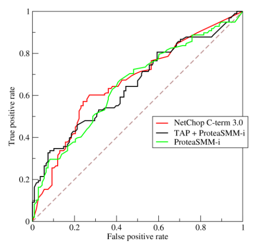


Figure 22: ROC Curve Wiki

Viola-Jones Filter

Viola Wiki, Viola Paper

Paul Viola + Michael Jones developed an object detection framework using ROC curves and computer vision. What they developed was a face detector that took a large dataset of faces which were centred + aligned. They used this data set to train a classifier. From this, if an image was fed in to the detector, it would put a box around what it perceived to be "faces". This was one of the earliest instances of a face detector. This is the gold standard face detector.

As we can see, it was a decent attempt, despite the football apparently being a face and some others being missed! How does the underlying code work and what does this have to do with ROC curves? Idea:

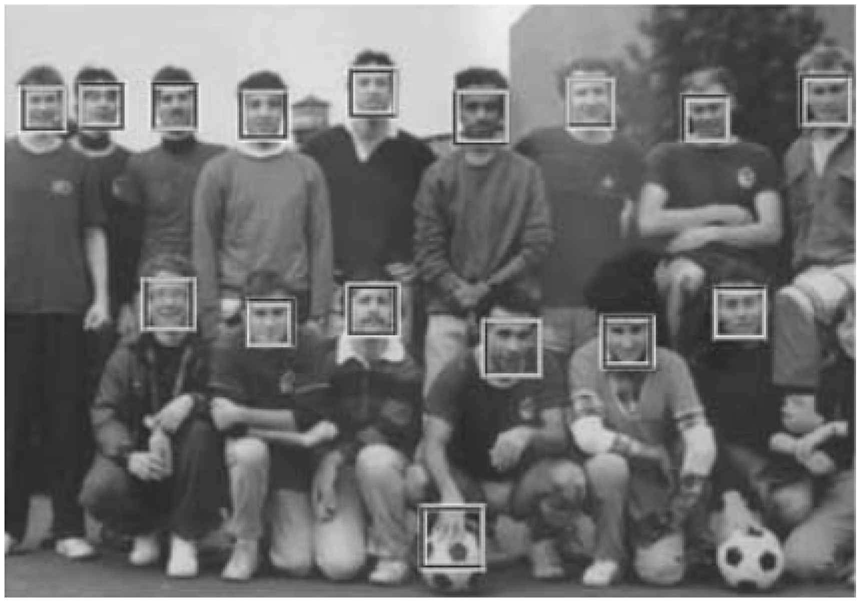


Figure 23: Viola Jones Example

- Make a crappy detector with a low false negative rate but high false positive rate.
- Essentially all "faces" this detector finds will be guaranteed to be complete. It will find lots of other non-faces too of course.
- We first apply this to the image because it is cheap and easy (otherwise we would need to take every possible 32x32 block of pixels in the image and see if it was a face)
- Then we can scan the gold standard algorithm over the areas marked as faces to cut out non-faces in most cases.
- This cuts out a lot of redundant work, saves a lot of time and cost and generally gives strong results!

Lecture 23

Reinforcement Learning

https://en.wikipedia.org/wiki/Reinforcement_learning

In supervised learning we have assumed that there is a target output value for each input value. However, in many situations, there is less detailed information available. In extreme situations, there is only a single bit of information after a long sequence of inputs telling whether the output is right or wrong. Reinforcement learning is one method developed to deal with such situations.

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment (world) so as to maximize some notion of cumulative reward. The agent (for example, it could be a robot of some sort) has to try and figure out the world and perform actions in order to get rewards (and minimise any penalties). Actions one performs now may affect the state of the world, and therefore any actions in the future as well. Reinforcement learning differs from standard supervised learning in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. Further, there is a focus on on-line performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). This is known as the exploration vs. exploitation trade-off.

The basic reinforcement learning model consists of:

1. a set of environment states S;
2. a set of actions A;
3. rules of transitioning between states;
4. rules that determine the reward of a transition
5. rules that describe what the agent observes.

A policy pi maps a State to an Action -> pi : S -> A

Temporal Credit Assignment Problem:

A network designed to play chess would receive a reinforcement signal (win or lose) after a long sequence of moves. The question that arises is: How do we assign credit or blame individually to each move in a sequence that leads to an eventual victory or loss? This is called the temporal credit assignment problem. With it, one basically wants to find out what action caused the reward/penalty.

The Method of Temporal Differences

https://en.wikipedia.org/wiki/Temporal_difference_learning

Temporal difference (TD) learning is a prediction-based machine learning method. It has primarily been used for the reinforcement learning problem, and is said to be "a combination of Monte Carlo ideas and dynamic programming (DP) ideas."

As a prediction method, TD learning considers that subsequent predictions are often correlated in some sense. In standard supervised predictive learning, one learns only from actually observed values: A prediction is made, and when the observation is available, the prediction is adjusted to better match the observation

The following is an example of when one could use Temporal difference learning:

Suppose you wishes to predict the weather for Saturday, and you have some model that predicts Saturday's weather, given the weather of each day in the week. In the standard case, you would wait until Saturday and then adjust all your models. However, when it is, for example, Friday, you should have a pretty good idea of what the weather would be on Saturday - and thus be able to change, say, Monday's model before Saturday arrives.

TD Gammon

https://en.wikipedia.org/wiki/TD-Gammon

One can train an artificial neural net with a form of temporal-difference learning in order for it to be able to play Backgammon.

***The* RL Book**

http://www.cs.ualberta.ca/~sutton/book/the-book.html

Nice Dated RL Survey

https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/rl-survey.html

Policy-Value Iteration

https://en.wikipedia.org/wiki/Markov_decision_process

Markov decision processes (MDPs) provide a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying a wide range of optimization problems solved via reinforcement learning.

Value iteration and Policy iteration are two notable variants of MDPs.

Q-Learning

https://en.wikipedia.org/wiki/Q-learning

Q-learning is a model-free reinforcement learning technique. Specifically, Q-learning can be used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP). It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. A policy is a rule that the agent follows in selecting actions, given the state it is in. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state. One of the strengths of Q-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment. Additionally, Q-learning can handle problems with stochastic transitions and rewards, without requiring any adaptations. It has been proven that for any finite MDP, Q-learning eventually finds an optimal policy, in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable.

Lecture 24

Recurrent and Deep Neural Networks

Recurrent neural network

Attractor network

Long short-term memory

Gradient calculation for dynamic recurrent neural networks: a survey