PHYS 498

Aiden Sirotkine

Spring 2025

Contents

Chapter 1

PHYS498

a whoooole lotta data analysis and fun stuff like that.

1.1 Clustering

group together sample data points that are a certain distance from each other

$$d(i,k) = \sum_{\text{features } i} (x_{ji} - x_{ki})^2$$

However, what if data points have different units?

ML algorithms are "unit-agnostic", which means they don't really care about units.

1.2 Whitening Transformation

$$x \to \frac{x - \mu}{\sigma}$$

it makes the mean 0 and the standard deviation 1

It just removes the white noise from a dataset.

a whitening transformation is a linear transformation that transforms a vector of random variables with a known covariance matrix into a set of new varariables whose covariance is the identity matrix, meaning that they are uncorrelated and each have unit variance.

whitening the inputs is useful because machine learning likes standardized data.

1.3 Examples

you slam bags of apples together and columnated jets of walnuts come out

1.3.1 Atlas

We look at the random jets of energy and we cluster the data to figure out what particles are where.

1.3.2 Gamma Ray Bursts

We can look into the universe and see random bursts of gamma rays

We find clusters of bursts at the galactic plane and around the center of the galaxy.

We cluster the data and discover the Fermi Bubble.

They're produced by colliding neutron stars and collapsing normal stars into black holes.

1.4 K-means Clustering

fast and robust decent algorithm. Assume you data consists of roughly round clusters of roughly the same size.

a_fit = cluster.KMeans(n_clusters=2).fit(a_data)

1.5 Hyperparameters

parameters that have to be pre-set that determine how the data is going to be analyzed.

For example, the number of clusters that we want as a result from K-means

1.6 ML Algorithms

Maximize the goal functions given the data.

The goal function $\mathcal{L}$ of the KMeans algorithm is

$$\mathcal{L}(c_j) = \sum_{i=1}^{n} \sum_{c_j=i} |x_j = \mu_i|^2$$

where $c_j = 1$ if the sample j is assigned to cluster I or otherwise

$c_j = 0$ and

$$\mu_i = \sum_{c_j=i} x_j$$

1.6.1 Expectation-Maximization

real important for alot of algorithms but I don't fully understand it.

1.7 Curse of Dimensionality

r^D is the volume of a D-dimensional hypercube with each dimension subdivided by r partitions.

a 3×3 rubix cube has 27 mini cubes in it because 3^3

It basically means that the more dimension you have, the more cooked you are to process all of it.

The curse of dimensionality.

If we have 30 dimensional data, we need over 1 billion data points in order to get a sample in each partition.

If there's a functional dependence between demensions, you can use a transformation to get rid of it and then you're working with less dimensions which is a win.

In the slides, 500 dimensional data can get transformed into 2 dimensional data.

1.8 Linear Decompositions

solve the eigenvector eigenvalue problem and delete the unimportant vectors and boom removed the most useless dimensions.

Principle Component Analysis (PCA) was developed in 1901 and is basically just removing the smallest/least impactful eigvenvectors

1.8.1 Covariance Matrix

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

is an estimate of the true covariance matrix using the data X comprised of N samples that are D-dimensional.

M is matrix where each row is an eigenvector

Y is a matrix such that $X = YM$

Remove dimensions from D to d with the smallest eigenvalues.

Now you have a much smaller matrix with most of the same data.

1.8.2 Eigenmatrix of $X^T X$

$$M^T = M^{-1} \qquad MM^T = I$$

$$X^T X = M^T \Lambda M$$

Where Λ is the diagonal matrix of decreasing eigenvalues.

The resulting latent variables are not correlated to each other, which means

$$\rho(j,k) = \frac{Y_J \cdot Y_k}{|Y_j||Y_k|} \simeq 0$$

1.9 Factor Analysis

another good way to reduce data

1.9.1 Non-negative Matrix Factorization

another thing

1.10 Independent Component Analysis

another thing

These aren't really important but like depending on what data you're doing you might need more than PCA.

1.11 Kernel Functions

If you add a bunch of random ass dimensions to your data, you can observe a number of correlations that you would not have

seen otherwise

$$\phi(x_0, x_1) = \begin{bmatrix} x_0^2 \\ x_0 x_1 \\ x_1 x_0 \\ x_1^2 \\ \sqrt{2c} x_0 \\ \sqrt{2c} x_1 \\ c \end{bmatrix}$$

This kernel function yields shenanigans

$$\phi(X_i) \cdot \phi(X_j) = (X_i + X_j + c)^2$$

This allows you to embed your data into higher dimensional space without actually using a bunch of math.

Our kernel function will be written as

$$K(X_i, X_j) = \phi(X_i) \cdot \phi(X_j)$$

A kernel function is a similarity measure, since it measures the similarity between samples i and j, with identical values being maximal and orthogonal values being minimal.

This is known as the kernel trick.

Sadly there are not an infinite number of kernel functions K

$$K(X_i, X_j) = (\gamma X_i \cdot X_j + c)^d$$

is called the polynomial kernel. There's also a sigmoid kernel and an infinite dimensional kernel

$$K(X_i, X_j) = \exp\big(\gamma |X_i - X_j|^2\big)$$

Because of the taylor expansion of e^x, this kernel function yields an infinite dimensional expansion.

1.11.1 Kernel PCA

goated method, but the data cannot be reconstructed super easily.

It uses the kernel function to expand the data to a bajillion features, and it then uses PCA to take out only the best features.

However, you need your hyperparameters to not suck.

1.12 Local Linear Embedding

Developed in the year 2000 which is more recent than most of the stuff we've done lol

a type of *Manifold Learning* (another way to reduce dimensions for non-linear data)

The sample is basically linear if you don't go too far away, so we can use a local linear approximation

$$\vec{X}_i \simeq \sum_{j \neq i} W_{ij} X_j$$

and we find that matrix W using a minimizing function

$$\sum_i |\vec{X}_i - \sum_{j \neq i} W_{ij} X_j|^2$$

We've now found a set of weights that described the non-linear geometry of the sample data

This geometry can be transferred to another sample space using another minimizing function

$$\sum_i |\vec{Y}_i - \sum_{j \neq i} W_{ij} Y_j|^2$$

Chapter 2

Probability Theory and Density Estimation

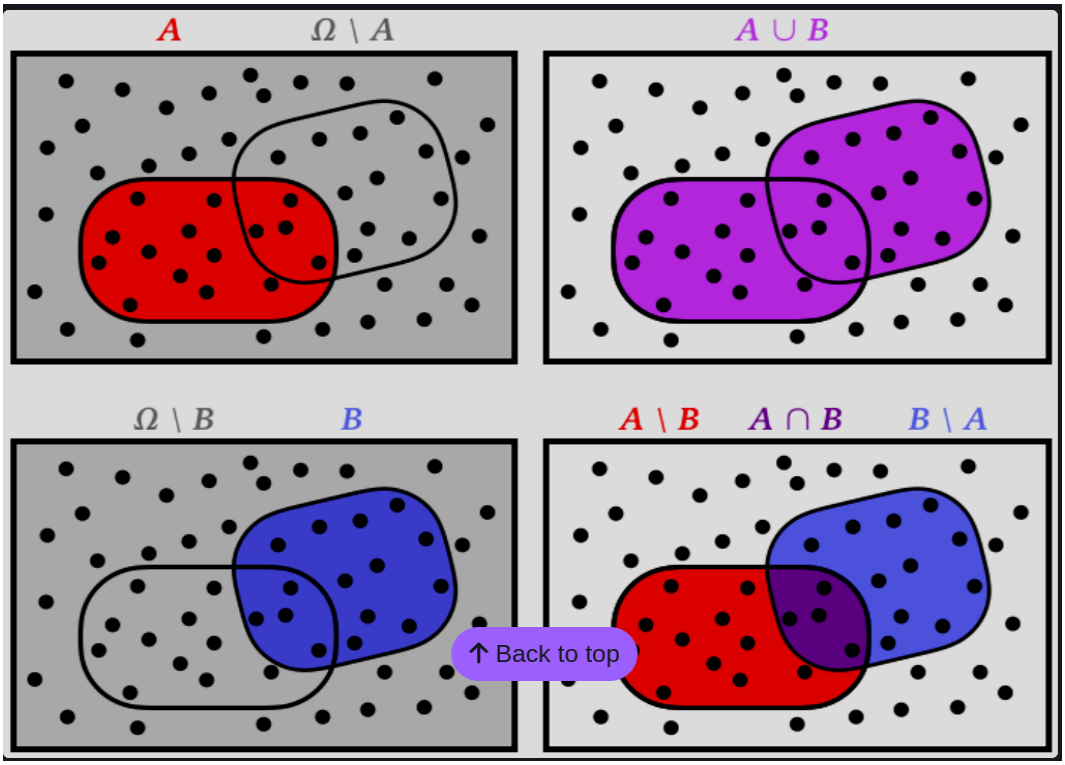
2.1 Axioms of Probability

1. A sample space Ω that defines the set of all possible uncertain outcomes

2. An event space $\mathcal{F}$ of combinations of outcomes (subsets of Ω)

3. A probability measure $P : \mathcal{F} \to [0, 1]$ that assigns numerical probabilities to each event.

   The tuple $(\Omega, \mathcal{F}, P)$ is a probability space



An event space must satisfy the following conditions

- If event A is in the event space, then so is its complement Ω/A
- If events A_1 and A_2 are included, then so is their union $A_1 \cup A_0$

2.1.1 Kolmogorov Axioms

- For any event A, $P(A) \geq 0$
- $P(\Omega) = 1$
- if all events have no outcomes in common, then

$$P(A_1 \cup A_2 \ldots) = P(A_1) + P(A_2) \ldots$$

2.1.2 Bayes' Rule

The probability of A given B

$$P(A|B) \equiv \frac{P(A \cap B)}{P(B)}$$

If A and B are independent, then

$$P(A|B) = P(A)$$

and then

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

Bayes' Theorem comes in handy for the disease test false positive shenanigans.

2.2 Random Variables

Let a random variable $X : \Omega \to \mathrm{R}$ labels each possible outcome $\omega \in \Omega$ with a real number $x = X(\omega)$

The probability is defined to be

$$P(X = x) \equiv P(\{\omega : X(\omega) = \omega\})$$

2.3 Cumulative Distribution Function

$$F_X(x) \equiv P(\{\omega : X(\omega) \leq x\})$$

It rises monotonically from 0 to 1 and is always well defined

2.4 Probability Density Function

$$f_X(x) \equiv \frac{d}{dx} F_X(x)$$

A PDF is a density function in the sense that

$$P(\{\omega x \leq X \leq x + \Delta x\}) \simeq f_X(x)\Delta x$$

2.4.1 Kernel Density Function

It's something

2.4.2 Gaussian Mixture Model

It's another algorithm that's good at grouping and density estimation.

2.5 Monte Carlo Method

Instead of integrating over a range, you can just take a random sampling to find the average over a function

$$\langle g \rangle \equiv \iint dx\, dy\, g(x,y) P(x,y)$$

We can use the standard deviations of x and y to find the correlation between the two

$$\sigma_x^2 = \langle (x - \overline{x})^2 \rangle$$
$$Corr_{xy} = \langle ((x - \overline{x})(y - \overline{y})) \rangle$$

You can use that to find the correlation coefficient

$$\rho \equiv \frac{Corr_{xy}}{\sigma_x \sigma_y}$$

2.5.1 Covariance Matrices

Useful

Chapter 3

Probability

3.1 Jensen's Inequality

Convex functions are nice because you can find the global maximum very easily.

$$g(\langle \vec{x} \ldots \rangle) \leq \langle g(\vec{x}) \rangle$$

The function at the expected value is always less than or equal to the actual expectation value of the function.

3.2 Probability Interpretations

3.2.1 Frequentist

Frequentist statistics means that the probability of an event is determined by the law of large numbers (the ratio of successes and failures will reach the probability with enough trials)

This is flawed for non-repeatable experiments (2016 NBA Finals)

3.3 Bayesian

You just like believe something has a certain chance based on past data.

Consider a joint probability distribution

$$P(D, \Theta_M, M)$$

with data features D, paramethers Θ_M, and hyperparameters M.

If we consider like the spins of subatomic particles, we have to consider the possibility that an event occurs even if it has never occured before.

3.3.1 Likelihood

Returns the probability density of observing x given parameters and hyperparameters

$$\mathcal{L}_M(\Theta_M, \vec{x}) = \sum_{k=1}^{K} \omega_k G(\vec{x}; \mu_k, C_k)$$

with parameters ω, μ, C.

3.3.2 Bayesian Inference

$$P(\Theta_M|D, M) = \frac{P(D|\Theta_M, M)P(\Theta_M|M)}{P(D|M)}$$

posterior = likelihood * prior / evidence

prior knowledge and new information $\rightarrow$ new knowledge.

3.3.3 Dice

Supposed you are given 3 dice rolls and have to guess how many sides are on the rolled dice.

The options are d4, d6, d8, d12, d20.

The dice rolls 6, 5, 4

It's definitely not a 4 sided die

If we guess the rolls are close to the mean, then its probably not 20 sided.

You can do some math and show that the most likely dice that was rolled was a d6.

3.3.4 Prior Probability

If your posterior data is greatly affected by your prior data, then you need more prior data.

A decent experiment's worth of data should be strong regardless of prior data.

You can also use an integral to compute Baye's rule over a

range

$$P(D|M) = \int d\Theta'_M P(D|\Theta'_M, M) P(\Theta'_M|M)$$

3.4 Graphical Models

graph CS not graph cartesian

Big probabilities can be turned into smaller joint probabilities.

$$P(D, \alpha, \beta) = P(D, \beta|\alpha)P(\alpha) = P(D|\alpha, \beta)P(\beta|\alpha)P(\alpha)$$

There's some pictures on the website

Chapter 4

Markov Chain Monte Carlo

Generate random samples from a non-normalized probability density

$$P(\vec{z}) = \frac{f(\vec{z})}{\int d\vec{z}\, f(\vec{z})}$$

You can find a half decent sampling using importance sampling

$$\langle g(\vec{z}) \rangle \equiv \int d\vec{z}\, g(\vec{z}) P(\vec{z}) = \frac{1}{N} \sum_{i=1}^{N} g(\vec{z}_i)$$

A random sampling of the MCMC data yields roughly the normalized integral

$$\frac{f(z)}{P(z)} = \int dz\, f(z)$$

4.1 Stochastic Processes and Markov-Chain Theory

Generates a sequence depending on all past data points in the sequence.

These processes end up making random distributions.

4.1.1 Markov Chain

It's a probability graph where the nth sample depends only on the $(n-1)$th sample.

It's a stochastic process with an extremely short term memory.

After enough time, the correlation between a stochastic process and its initial conditions decreases.

We talked about these guys in lin-alg they're pretty cool.

4.1.2 Stationary

The update rule is always the same

The probability of the M2 given M1 is the same as M3 given M2

so you can iterate forever

Markov chains reach an equilibrium after a while based on some eigenvectors.

After a bajillion iterations for stationary markov chains, the dependence on the initial values entirely disappears.

Works for all stationary markov chains.

You get a probability density

Markov chains are reversible if there symmetry along the diagonal axis $y = x$

4.1.3 Custom Markov Chains

Metropolis-Hastings-Green algorithm is an algorithm for making a markov chain with a pre-determined probability density.

4.1.4 Markov-Hastings

It relies on a proposal distribution that is easier to sample from than the target distribution.

You can choose any proposal distribution, but you should choose one that is similar to your actual probability density to get to equilibrium faster.

There's a formula in the lecture notes.

4.1.5 Gibbs Sampling

another type of MH algorithm

If we want to sample a 3d distribution, we take 3 different samples of the conditional probabilities.

You end up getting something that is proportional to the true 3d sample

4.1.6 Hamiltonian Sampling

Calculate all partial derivatives of our target $\log\left(\tilde{P}\right)$

You do some Hamiltonian witchcraft that I probably would maybe understand if I took 325 and then you get a distribution.

Canonical Distribution.

4.2 Bayesian Model Selection

We remember Baye's Rule, but now we have to figure out what model we should use to get the best probabilities.

$$P(\Theta_M|D, M) = \frac{P(D|\Theta_M, M)P(\Theta_M|M)}{P(D|M)}$$

We need to figure out a model M to use

We have a number of methods that we can use that are all in the slides.

If we do pairwise comparision, then the probabilty of the evidence cancels out and we don't have to deal with it.

$$\text{Baye's Factor } = \frac{P(D|M_1)}{P(D|M_2)}$$

Bayesian inference is a fan of Occam's Razor: the simplest model without prior evidence is the most likely model.

We can imagine the same thing with 1 big gaussian or 2 small guassians being the differing models for a certain data set.

There's a Jeffrey's Scale that's basically just metric

100 is very strong M1, 10 is kinda likely M1, 0.1 is kinda likely M2, 0.01 is very strong M2.

Model is real involved you should come back to this.

4.3 Variational Inference

VI provides an exact description of an approximate posterior distribution (using optimization).

MCMC provides an approximate description of the exact posterior probability density using sampling.

4.3.1 Kullback-Leibler Divergence

It's a formula to determine how "close" two functions are to each other.

$$KL(q||p) = \int q \log\left(\frac{q}{p}\right)$$

It can also be considered the difference in the expected values between the logs of q and p

There's then more stuff

$$P(\theta) = P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

$$KL(q||p) = \ln(P(D)) + KL(q||P) - \int d\theta q(\theta) \ln(P(D|\theta))$$

and those last 2 terms are called ELBO for some reason.

$KL(q||P) - \int d\theta q(\theta) \ln(P(D|\theta))$ is the evidence lower bound (elbo)

4.3.2 Variational Inference Method

Take a sample function and minimize the KL divergence to find the sample function that most closely approximates our probability density.

You can calculate the KL divergence and find the minimums of certain functions analytically.

4.4 Evidence Lower Bound

remember Baye's rule

$$p(\theta) = \frac{P(D|\theta)p(\theta)}{P(D)}$$

The probability of the model is the probability of the data given the model times the probability of the data divided by the evidence.

4.4.1 Variational Bayesian Inference

define a family of functions that could approximate the posterior probability density.

Use optimization to find the best function

$$P(\theta) = P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

$$\int d\theta\, q(\theta) \left(\ln(P(D)) + \ln\left(\frac{q(\theta)}{P(\theta)}\right) - \ln(P(D|\theta)) \right)$$

$$KL(q||p) = \ln(P(D)) + KL(q||P) - \int d\theta q(\theta) \ln(P(D|\theta))$$

and those last 2 terms are called ELBO for some reason.

$KL(q||P) - \int d\theta q(\theta) \ln(P(D|\theta))$ is the evidence lower bound (elbo)

- The three terms are the log of the evidence $P(D)$,
- THe KL divergence of $q(\theta)$ with respect to the prior
- and the q-weighted log-likelihood of the data

The log of the evidence is a constant offset in the sum
the KL divergence is minimized when $q(\theta) = P(\theta)$
the log-likelihood is minimized when q prefers parameters that explain the data

$$\ln(P(D)) = \int d\theta q(\theta) \ln(P(D|\theta)) - KL(q||P) - KL(q||p)$$

$$\ln(P(D)) \geq \int d\theta q(\theta) \ln(P(D|\theta)) - KL(q||P)$$

$$ELBO \equiv \int d\theta q(\theta) \ln(P(D|\theta)) - KL(q||P)$$

Using this, we also find that

$$KL(q||p) = \ln(P(D)) - ELBO(q)$$

The evidence based lower bound can be written as the sum of a bunch of expectation values

$$ELBO(q) = \langle \ln(P(D|\theta)) \rangle_q + \langle \ln(P(\theta)) \rangle_q + \langle \ln(q) \rangle$$

4.4.2 Example (Laplacian PDF)

the probability of observing x given some model parameter θ is

$$P(x|\theta) = \frac{1}{2} e^{-|x-\theta|}$$

So our resuling likelihood is

$$P(D|\theta) = \prod_i P(x_i|\theta)$$

Let our prior knowledge be specified by a unit Gaussian (approximate function)

so our posterior probability density function is

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Now we let our approximate function be a unit Gaussian to optimize.

4.4.3 Practical Applications

Markov Chain Monte Carlo always gives you something as long as you have a likelihood and prior.

VI is generally more computationally efficient, but takes a little more to set up.

It takes some amount of judgment and prior knowledge to have a decent approximating function q

4.5 Optimization

real important math

$$x^* = \text{argmin} f(x)$$

The easiest way to approximate the minimum is just sample over the entire function and take the smallest result, but this is not very accurate. If you sample distances are too big, you'll miss details

4.5.1 Gradient Descent

make constant small steps in the direction perpendicular to the gradient

η is the learning rate and determines how big of jumps you make per step.

You can find the gradient by calculating derivatives numerically

$$\frac{\partial}{\partial x_i} f(x) = \frac{f(x + \delta e_i) - f(x - \delta e_i)}{2\delta} + \mathcal{O}(\delta^2)$$

Very fast, but not super accurate

4.5.2 Auto-differentiation

a hybrid between the difference equations and an analytical approach

You take a number of primitive functions and you numerical factors to optimize those primitives to your real function.

It's extremely fast and extremely accurate.

Doesn't work for diverging points

If you just put a value there then everything works.

4.5.3 Optimization in Machine Learning

K-means clustering is just an optimization function that optimizes

$$\sum_{i=1}^{n} \sum_{c_j=1} |x_j - \mu_i|^2$$

Optimization is also useful in Bayesian Inference.

consider the maximum a-posteriori (MAP) point estimate

$$MAP \equiv \text{argmin}_\theta(-\ln(P(\theta|D)))$$

You can see the MCMC calcualtion with the MAP estimate
You can also find the maximum likelihood (ML) point

$$ML \equiv \text{argmin}_\theta(-\ln(P(D|\theta)))$$

You also see optimization is Variational Inference

$$VI \equiv \text{argmin}_\lambda(-ELBO(q(\theta;\lambda)||P(\theta|D)))$$

4.5.4 Optimization Methods

There are a bunch but just use the best one which is stochastic gradient descent.

People benchmark their optimization functions based off of the Rosenbrock function.

You use automatic-differentiation to get the gradient and then you use gradient descent to optimize the function and boom

4.5.5 Stochastic Gradient Descent

gradient descent but use only a small sample subset of the function, called a minibatch.

It takes more steps, but drastically reduces the amount of data necessary.

the noise caused by SGD helps prevent overtraining by adding noise to the data.

4.6 Cross Validation

There are 3 types of learning

- learning model parameters from data
- Learning to predict new data (unsupervised learning)
- Learning to predict target features in new data (supervised learning)

All three of these models require previous data to do anything.

Now we have to figure out how to compare models to learn which is best.

Bayesian evidence is the main tool to see which model is preferred by the data.

The way you do this is with multiple different datasets

4.6.1 Training Set

This is how you initially train the model you make your initial curve fitting parameters.

4.6.2 Validation Dataset

After the model has been trained, you evaluate the model based on new data, but you still tune the model's hyperparameters while it looks at the validation set.

Mainly for fine tuning. Hopefully you don't need to make major changes after the model has gone through the training data.

YOU CAN ONLY CHANGE THE HYPERPARAMETERS, YOU CAN'T CHANGE THE MAIN PARAMETERS.

4.6.3 Test Dataset

Once the algorithm is completely trained, you give it unseen data as a final test to see if your model is any good in a real world scenario.

4.7 Cross Validation Process

We will study the *K-folds* method of cross validation.

4.7.1 Overfitting and Generalization

Competing models can be considered as P order polynomials with $P + 1$ parameters.

You can use linear regression to implement a fit and then make a pipeline (python library)

An overfit model will go over too many data points but not get the big picture.

An underfit model does not have enough parameters so it is also missing the big picture.

4.7.2 Train-Test Split

There's a python library that automatically cross-validates data.

Somewhere between 10% and 50% (sklearn automatically does 20) is a decent split for training data and testing data.

4.8 K-Folding

Split the data multiple times and combine the correlated results.

The more k-fold splits you make, the more correlated your data will be.

It's called K-Folding because you split the data to roughly k samples of roughly k datapoints.

It works pretty well but you have to check your polynomial degrees to prevent overfitting

4.9 Comparison with Bayesian Evidence

We can use MCMC to calculate Bayesian evidence.

Chapter 5

Artificial Intelligence

5.1 Intro

skip

5.2 Supervised Learning

Instead of doing Bayesian evidence probability stuff, supervised learning is trying to map data to results and eventually get a function that acts as a predictor (hypothesis).

A regression problem maps stuff to stuff continuously.

A classification problem maps all stuff to some discrete number of things.

5.2.1 Linear Regression

Matrix math to make a best-fit line.

There's some probability stuff and some error math but I missed it

5.2.2 Linear Deconvolution

We are trying to find the response function of the system.

The response function is function such that

$$z'(t) = \int dt' \, R(t - t') z_{input}(t')$$

This is called a convolution.

So we're going to use a linear model to try and deconvolute the convolution.

The issue is when there's any noise in your model, the deconvolution becomes horrendous.

Deconvolution amplifies noise.

5.2.3 Regularization

Use fancy math to minimize the noise in a model so that deconvolution actually works well.

Chapter 6

Artificial Intelligence

AI is a computer doing human-level reasoning

ML is a computer learning without explicitly being programmed.

Artificial Neural Networks have node and hidden layers and whatnot

Deep Learning is an ANN with more than 3 layers.

You have a number of different types of learning

- Supervised
- Unsupervised
- Semi-supervised
- Reinforcement

Supervised Learning is defined by how data is labeled.

Unsupervised learning just recognizes patterns

Deep Reinforcement learning is very very time consuming and requires a whole lot of data.

inputs go into a black box that makes outputs.

6.1 Neural Network

Generic, flexible, trainable, modular, and efficient.

It's pretty good and can map most non-linear functions.

Given some data, you mulitply it by a weight and then plug it into some activation function.

6.1.1 Network Layer

Consider the weight as a matrix instead of just a constant.

There are all sorts of different decision boundaries depending on your activation function.

We take a tensor of input data and it goes through a number of weighted matrix multiplications until we get some output tensor.

6.1.2 Tuning

You can look at the results of the function.

If we calculate the mean squared error and set its derivative to 0 then we can do some math to tune our ML model.

6.2 Pytorch

It allows you to make an ML model pretty easily.

It randomly makes the weights.

6.3 Loss Functions

This is really the thing that Gradient Descent is talking about

$$\theta \to \theta_i - \eta \frac{\partial l}{\partial \theta}$$

Wherer l is your loss function.

You find a loss function and then you calculate the gradient and then good things happen.

6.3.1 Regression Loss

This is the way that regression functions in lin alg work. It's basically just the squared error

$$L_2 = \frac{1}{2}|X_{out} - Y_{tgt}|^2$$

Minimizing the loss is the same as finding the maximum likelihood point estimate

6.3.2 Binary Classification Loss

The binary cross entropy is a better loss function for binary classification problems (cat or dog (or whatever))

The equation is in the thingy

most code uses either BCE or MSE loss.

What is MSE loss though?

6.3.3 Multicategory Classification Loss

This is another crazy loss function that uses some math to optimize for if there are multiple discrete categories that you are trying to match.

This is all related to Bayesian Variational Inference (VI)

The K-L Divergence when grouping stuff like a month ago is also related to the minimizing loss functions.

6.3.4 Deep Neural Networks

MLP - Multi-Layer Percepteron

You connect layers of nodes to new layers of nodes until something good happens.

You can drop out random neurons to test how robust the neural network out.

You can have nodes feed back into themselves (Recurrent NN)

You can have Auto Encoders and Variational Auto Encoders.

A GAN is a Generative Adversarial Network.

Neural Networks can approximate universal functions.

Deep learning just needs data instead of outside feature extraction.

6.3.5 Minibatch Gradient Descent

You batch your gradient descent steps and as long as your batches are small enough you reduce compute time without

reducing error.

6.4 Generalization

Avoid underfitting and overfitting

In underfitting, your validation and training error are about the same

In overfitting, you training data has very low error but your validation error is too high.

6.4.1 Drop Out

A good way to generalize your function is Drop Out

Just get rid of 20-50 % of your nodes

6.4.2 Early Stopping

watch how your validation error changes and stop when the validation error goes up.

6.5 Convolutional NNs

Mainly used for image data.

They use a convolutional operator to extract data features.

These NN's are robust against spatial translation (they can find the same features even if theyre in different parts of the image)

It takes larger image and turns it into less data using the convolutional function.

You can have multiple layers of convoluting and max pooling to turn a whole bunch of data into a little bit of feature data.

Far better for image processing than regular feed-foward networks because the convolution compresses the data.

6.5.1 Caveats

Data has to be put on a regular grid.

6.6 Recurrent NNs

Used for modelling sequential data (Language Models). Youre data is coming in a sequence.

It triggers recurring connections between neurons.

RNN's can be used for basically anything as long as there is ordered data.

Feed Forward NN's always move data from left to right.

Recurrent NN's also allow for data to be left inside the network as new data is being inputted, allowing for some time of stochastic shenanigans (LLM's)

Because recurrent NN's use looping, they use far less nodes than deep NN's and can be put on something like an FPGA.

6.6.1 Caveats

The order of the data now matters.

Input data needs to be packaged into variable-length messages

Gradient Descent is possible but involves "unrolling" the nodes.

6.7 Long-Short Term Memory Networks

You have an input gate, output gate, and a forget gate.

It limits the amount of information that can be passed from one cell to the next.

6.8 NN talks

We've had NN's for a while but they weren't useful until we thought about using non-linear activations functions (tanh, sigmoid)

We also use backpropogation to increase the strength of NN's

Skipping connections also made AI pretty good by increasing generaliztion.

Chapter 7

Deep Learning

It's real useful

7.1 Geometric Deep Learning

Non-Euclidean data can represent more complex informaition.

A non-euclidean datatype is a graph

molecules are realy good examples of information that should be represented with a graph.

Nodes in a graph can have features.

Tons of stuff can be represented in a graph (molecules, information, neurons, genes, communication networks, software)

7.2 Graph Setup

V is the vertex set

A is the adjecency matrix (assume binary)

$X \in \mathbb{R}^{|V| \times m}$ is a matrix of node features

v is a node in V, and has $N(v)$, which is the set of neighbors of v

7.3 Naive Approach

You just plug the adjacency matrix directly into the thingy.

There are $O(|V|)$ parameters.

It is not applicable to graphs of different sizes

It is sensitive to node ordering.

7.4 NN Invariants

Deep NN's can "learn" translation, scale, rotations given enough data through the changes and shared features.

7.5 Graph Neural Network

A GNN is a class of GDL methods for modeling data via message passing over graphs

GNN architectures ar edesigned to learn a nembedding that contains informaition about its neighborhood.

Relationships of increasingly distant nodes/edges incorporated iteratively through use of additional message passing steps.

GNNs learn contextual relationships between nodes

7.5.1 LHC Example

You have sensor at various distances, and you create a sparse graph of potential paths that the particles could have taken from where the sensors sensed a hit.

The GNN returns true or false on given edges, and the true edges are the most likely paths that particles took.

Once you train a GNN, you're able to determine the likely paths of particles extremely quickly without a ton of computer power.

We have a bunch of nice python code in the website to show how GNN's work.

7.5.2 Graph Convolutional Network

They're like GNN's except they also use a convolution function to reduce a full graph into important traits.

7.5.3 Deep Sets

It's a deep neural network library that receives samples of a discrete set of data.

There's a whole paper made in 2017 that talks about how it works.

7.6 Attention

From a paper called "Attention is All You Need".

It talks about transformer models and how attention is used to improve deep learning models.

Recurrent neural networks take in remembered data from previous samples.

You can make a bidirectional RNN that determines information at time t by looking at information both before and after t.

Causal RNN's only predict future data given past data.

The biggest issue with these RNN's is you eventually run out of memory.

7.6.1 Attention Augmented RNN

It essentially gets the weight average of the correlations between tokens to figure out what parts of a sequence matter the most.

This is related to convolutions because a convolution function summarizes a spread of data, and the attention function summarizes the past and present information.

We use the softmax algorithm to weight our external data.

You have QUERY, KEY, VALUE items and you have to find the similarities between QUERY and KEY.

You do some matrix math and good things happen.

Vectors that are closer together have larger dot projuctsmore similar vectors yield a greater attention.

Everything is matrix multiplication.

7.6.2 Enforcing Causality

You can force certain parts of the attention matrix to be 0 depending on how our sample data is correlated to itself.

7.6.3 No Learnable Parameters

You can literally just compute the autocorrelation XX^T

7.6.4 Crazy Amounts of Technicalities

Just read the website lecture notes.

7.6.5 Cross Attention

We have two different streams of data that relate to each other, and you need attention between streams of data.

7.6.6 Flash Attention

It's something to do with GPU programming

Chapter 8

Transformers

Transformers are the thing that revolutionized the world like 3 years ago.

They are very impactful in NLP (Natural Language Processing), but we won't go over that in crazy detail at the moment.

Transformers are AI architecture that uses attention to predict data.

Transformers are also very good at computer vision.

8.1 Attention DFN

The attention mechanism describes a weighted averag eof (sequence) elements with the weights weights dynamically computed based on an input query and elements' keys.

What transformers use mainly is self-attention.

8.1.1 Scaled Dot Product Attention

We want an attention mechanism where any element can affect any other element, but the mechanism is still efficient to compute.

You do some math and take the softmax of it.

8.1.2 Tilde

A tilde means to sample from a certain things

q_i $N(0, \sigma^2)$ means that q_i is sampled from a normal distributed centered around 0 with a standard deviation σ^2

8.2 Multi vs Single Headed Attention

Multi-Headed attention essentially means you get multiple matrices that each can affect the data according to the attention.

This allows you to look at multiple possible relationships between data.

8.3 Transformer Architecture

It's on the website, but the main thing that its just a lot of multi-headed attention blocks in a feed forward network.

Transformers utilize skip connections.

8.4 Positional Encoding

We let the multi-headed attention network understand the relative positions of its data encoding it via an input feature.

We do some math to let the computer know the ordering of things.

We are trying to break the equivariance of the multi-headed attention.

8.5 Learning Rate Warmup

We have to start the learning rate close to 0 and gradually bring the learning rate up to where we want it originally, and then it can go back down.

8.6 Multi-Headed Attention

I only added this to really cement it as the important part of transformers.

Transformers are special because they have multi-headed attention compared to other attention-augmented neural networks.

The weighting based on dot product similarity and

the batching of attention in transformers

The weighting is also important

Chapter 9

Generative AI

Machine learning is a part of Artificial Intelligence, and Generative Models are a part of Machine Learning, and Deep Generative Models are a part of Generative Models and Deep Learning.

9.1 Discriminative vs Generative Models

A discriminative model will, given a set of input data, distinguish (discriminate) that data between various traits.

Generative models can, given traits, make data that the discriminative AI will recognize as having that trait.

Discriminative models learn a probability distribution given data $p(y|x)$

Generative models learn a probability distribution itself $p(x)$

Generative models can learn features without labels.

Conditional Generative models can assign labels while rejecting outliers. It generates new data conditional on input labels.

The way that generative models work is by converting noise to real data.

9.2 Types of Generative Models

- GAN : Adversarial Training with a Discriminator
- VAE : Maximize variational lower bound with an Encoder
- flow based models : invertible transform of distributions
- Diffusion : Gradually add Gaussian noise and then reverse the noise

  The flow models uses invertible transforms and Jacobians.

9.3 Applications

Generative models are generally not better than full simulations of data, but they are much cheaper and much faster, which is their real benefit.

9.4 Auto-encoders

It turns data into less data while keeping the same information.

Specifically its a neural network that compresses data.

9.4.1 Mp4 Compresion

'It takes reference images, and instead of storing every single pixel, it will just store the change in pixels between each image. Now our filesize is very small, but our CPU needs to do more work to turn the compressed data into the actual video.

Autoencoders turn a bunch of redundant information into only the necessary information, with the rest being able to be extrapolated from an algorithm.

Autoencoders do unsupervised dimensionality reduction.

9.5 Curse of Dimensionality

We need Autoencoders because without them, data becomes extremely expensive to do statistics on and impossible to interpret.

9.5.1 PCA

It works great, but has some limitations

Auto-encoders are non-linear PCA, and the components DO NOT have to be orthogonal to each other.

9.6 Training

What your loss function is is the loss of information from encoding to decoding.

You want to decrease the amount of space in your data whilst losing as little information as possible after decoding the reduced data.

The way that you actually decrease the amount of data is by making a bottleneck (smaller and smaller layers) such that less information can go through each layer.

9.7 Convolutional Autoencoder

Instead of using regular matrix layers, you can put convolutional layers in an autoencoder to get dimensionality reduction.

9.8 Generation

You can use the decoding part of an autoencoder as a generative AI.

The issue with this is that the generational space is extremely large and not regularized, so sampling from it can yield absurd results.

9.9 Variational Autoencoder

- You marry the Variational Inference method with autoencoders to reguarize your inference space.
- You use the KL Divergence relative to a unit Gaussian. Basically we minimize the difference between our distribution and a standard Gaussian.

- What you do is you add an additional term to the loss function. The additional term requires that the latent space variables are unit Gaussians.
- Now our latent space is a multidimensional unit Gaussian.
- Our big unit Gaussian latent space is essentially our noise that we can sample from randomly to make image generation.

9.9.1 Generating Data

It works pretty well, you just sample noise from out big thing of Gaussians and decode it to get an answer.

The issue with variational autoencoders is that there is now a smooth transition between each sample, so the autoencoder will create a mix of every distinguishing trait instead of one at random.

You need to add some sort of discretization in order to make specific traits from a variational autoencoder.

Chapter 10

Generative Adversarial Networks

These are called GAN's

- The way that they work is in tandem with a discriminatory neural network.
- It essentially generates data until the discriminator NN says that the generated data has the desired trait.
- The discriminator is trained on real data, and tells the generative NN whether or not the generated data has the corrected traits.

10.1 Issues

The main thing with GAN's is they are are very finnicky, and they can yield very large losses if not trained correctly.

10.1.1 Mode Collapse

Sometimes a GAN will only generate 1 type of output or a small subset of outputs.

If the discriminator gets stuck in a local minimum, then the generator can repeat the same output and not get trained successfully.

10.1.2 Vanishing Gradients

When the discriminator is only slightly better than the generator, the update weights will disappear and it will stop improving.

10.1.3 Convergence

The GAN's effectiveness will oscillate because the discriminator is so bad that it is essentially a coin flip.

10.2 Conditional GAN

It allows the user to not only generate images, but to also label them with specific traits.

Chapter 11

Diffusion

- You make a de-noising machine, and then you give it all noise and it tries to generate an output.
- Adding noise is easy, removing noise is hard
- The noise is not specifically random, but is developed from a noise scheduler.
- The noise is always Gaussian.

11.1 Implementation

You actually use Variational Inference to make the de-noiser because the denoising algorithm itself involves a 1000 dimensional integration.

11.2 Normalizing Flows

This is a mathematical method that turns a simple probability distribution to a more complex multimodal distribution.

You transform the simple distribution into a more complex one by using a sequence of invertible nonlinear transforms.

You use neural networks to figure out the invertible transformations/mappings that work best to change the distribution.

11.2.1 Benefits

Other NN's do not explicitly learn the probability distribution.

Variational Autoencoders only model and minimize the Evidence Based Lower Bound (ELBO), and GAN's generate new data without telling us where on the probability distribution that data lies.

Because all the transformations are invertible, the math for the NN is shockingly simple compared to every other NN we've looked at.

11.2.2 Different Types of Flows

Planar flows are also a type of transform that is commonly used.

It is invertible and fast to compute.

Idk if the NN optimizes each transform sequencially or all at the same time

Chapter 12

Simulation Based Inference

Also called LFI

It allows you to figure out something (make in inference) even if it has an intractable likelihood.

Approximate Bayesian Computation is another thing worth looking at.

We used a thing called a bump hunt to figure out the Higgs Boson.

12.1 Explicit Likelihood Calculator

You can use a Poisson Distribution over the mean returned by the forward model to realize the explicit likelihood.

12.2 Implicit Likelihood

You can figure out a likelihood without explicit calculation using the

12.3 Approximate Bayesian Computation

The idea behind Approximate Bayesian Computation (ABC) is to realize samples from the forward model (with the parameters drawn from a prior) and compare it to the dataset of interest. If the data and realized samples are close enough to each other according to some criterion, we keep the parameter points.

12.4 Neural Likelihood Ratio Estimation

12.5 Explicit Likelihood

12.6 Test of Statistical Coverage

Chapter 13

Reinforcement Learning

Instead of labelling all of your data, there is seeding that will guide how the NN acts depending on its reponses.

It's a robot that maximizes some reward function.

Supervised learning has the goal already connected to the datapoints.

Reward(i), State(i + 1) = Interact(State, Action)

And the action can be tweaked depending on the subsequent rewards.

13.1 Temporal Difference (TD) Learning

a class of model-free reinforcement learning methods.

There was a back-gammon Reinforcement Learning NN that learned by just playing against itself and maximizing the reward function.

Reinforcement Learning is real good at learning to play videogames.

It also works for locomotion.

Example: AlphaGo

13.2 Deep Q-Learning Network (DQN)

This is just the name of a prototypical Reinforcement Learning NN

13.2.1 Setup

There a Markov Decision Process with time steps where the agent makes some action to change the environment.

The value of a new state is based off of a value function.

$$V^{\pi}(s) = \sum_{s'} P_{\pi(s)}(s, s') * (R_{\pi(s)}(s, s') + \gamma V^{\pi}(s'))$$

Where s is the current state and s' is the new state.

13.2.2 Q Learning

Learn a policy $\pi(s)$ that optimizes $V^{\pi}(s)$ for all states, using no prior knowledge of either the state transition probabilities or the reward function.

Take random actions until something good happens.

13.3 Challenges

- If there are too many states and actions, then your NN will wander around and rarely ever change, so it improves reaaaally slowly.
- If the episode can end without reward, then your NN will not change
- If there is a narrow path to the reward, then there is a low change your NN will find it and improve

13.3.1 Reward Shaping

Instead of just having the big reward at the end, you should place a bunch of little bread crumbs to help the NN learn with a little big of guidance.

13.3.2 Memory

You can retrain on previous explorations instead of running a bajillion more simulations.

It allows you to leverage each episode as much as possible if it is hard to get the reward.

13.4 Gym

It's a toolkit for reinforcement learning so you don't have to do all the silly math yourself.

13.5 Policy Gradients

You get a game-state / frame

If you win, you make all of your actions more likely.

If you lose, you make all of your actions less likely.

Chapter 14

AI Explainability and Uncertainty Quantification

Deep Learning networks get better and better with more data no matter what.

Tradition machine learning saturates after a bit.

Deep Learning saturates much much later, and only gets better with more layers.

Deep Learning has a large number of parameters and they use non-linear representations.

Deep Learning takes an incomprehensible amount of data and layers to understand intuitively. The ATLAS Software might as well be magic.

14.1 Benefits

Explainability allows us to improve NN's and make them more robust against bias or noise or whatever.

Explainable AI is hard to define and even harder to evaluate

XAI is important if you AI is actually doing something useful compared to like game design or whatever.

14.2 Tools

Simpler AI methods like linear regression are highly explainable.

There's a bunch of tools that work and none of them are crazy better than the others.

14.2.1 Performance Deviation

Very fast, but not super accurate.

It just goes feature by feature and looks at what affects the results.

14.3 Layerwise Relevance Propogation

You're back-propogating through the network and figuring out the importance of each node relative to the entire model.

14.3.1 TopoDNN

This is a NN that the prof made that was able to distinguish particles from their jet tracks.

14.4 Neuron Activation Pattern

It's like doing an MRI of the NN.

You basically look at what neurons do what when.

14.5 PCA

You can make a PFN Latent Space that distinguishes that largest traits that the NN also sees.

14.6 Probabilistic Learning

If the input is unlike anything during training, the output will be nonsense.

14.6.1 Aleatoric Uncertainty

Describes the confidence in the input data Large when input data is noisy

Cannot be reduced by simply adding more data.

This can be learned directly using neural networks.

14.6.2 Epistemic Uncertainty

This is the model uncertainty.

Describes the confidence in the prediction.

Large when insufficent training data

Can be reduced by adding more data.

This is hard to estimate.

14.7 Bayesian NN

It's a neural network that solves a Bayesian Inference problem.

These types of problems are not tractable and need approximations.

There are a number of sampling methods that can be used to approximate the NN (Monte Carlo Dropout)

Can we just learn the probability distribution of the sample data?

14.8 Evidential Deep Learning

treats learning as evidence acquisition, and more data leads to a greater confidence.

14.8.1 Conjugate Priors (IMPORTANT CONCEPT)

Choice of evidential distributions is closely related to conjugate priors in the context of Bayesian inference

14.8.2 Training

Train the network to output the parameters of an evidential distribtuion.

Perform multi-objective training.

14.9 Conformal Prediction

it allows us to construct statistically rigorous uncertainty bands around our predictions, without requiring any modifications to our prediction model. This is achieved by comparing true and predicted values on out-of-sample data (more precisely we are looking at inductive conformal prediction)

14.10 Misc AI Shenanigans

Quantum Computing, AI Bias, General AI

Chapter 15

Unsupervised Learning and Anomaly Detection

Supervised learning is just your regular classification structure that has "answers" for each individual datapoint.

Unsupervised learning just finds patterns in the data that you give it. It finds structure and learns how certain samples are the same/different to other samples.

The main benefit of unsupervised learning is that it can find hidden patterns that humans have missed.

There are two main types of unsupervised learning: Clustering and Dimensionality Reduction

Both are just ways of finding patterns in large datasets

15.0.1 Clustering

Divide the dataset into groups

15.0.2 Partitioning Clustering

This is the clustering that we discussed before with KMeans
Divides the data into non-hierarchical groups

15.0.3 Density Based Clustering

Highly dense areas are put into clusters

This method is good because it can make arbitrary shaped clusters

15.0.4 Distribution Based Clustering

This uses the Expectation Maximization algorithm, but for Gaussian Mixture Models, which makes it different to KMeans

15.0.5 Hierarchical Clustering

The clustering is tree-like, where one datapoint can be in multiple clusters. This is called a dendrogram

The most common algorithm is called the agglomerative Hierarchical algorithm.

15.0.6 Fuzzy Clustering

It is similar to KMeans clustering, but datapoints can be in multiple clusters, so its called "soft" clustering.

15.0.7 Dimensionality Reduction

Get rid of data that is not helpful to the actual information of the dataset.

15.1 Training Paradigms

Specifically in high energy physics

You can use supervised learning and data from simulations

15.1.1 Semi-supervised

Use a small amount of labeled data and a large amount of unlabeled data

15.1.2 Weakly-supervised

use noisy or imprecise sources to label the data.

In your labels, have a small fraction of labels be wrong

15.1.3 Self-supervised

take advantage of the underlying structure in the data to label it instead of labeling it externally.

Chapter 16

Anomaly Detection

16.1 Autoencoders

Compress the data into a smaller latent space and then decode it back to the original state.

If the event is an anomaly the reconstruction loss can be used as an anomaly score.

16.1.1 LSTM

Long Short-Term Memory

It's a type of RNN good for handing long-term dependencies

For an AE or VAE (variational autoencoder), you should have a higher latent space compared to if you want it to be an anomaly detector.

Anomaly detectors do not need high latent spaces because they do not need to accurately reconstruct the data, just

LSTM RNN's utilize memory cells to store long-term infor-

mation and gates to control the flow of memory.

LSTM's are especially good at mitigating vanishing gradients.

The prof is also looking at Evidential Deep Learning for anomaly detection.

16.2 Anomalies

An anomaly is a discovery returned from an AI that was not observed by humans, so they're pretty neat.

However, most of the time, anomalies are just bugs in the AI.