Classical Machine Learning

Introduction

Observable quantity, x related to some parameters, θ
↳ Model: $p(x|\theta)$ - probability x given θ

Perform experiment to obtain data set X
↳ Use this data to fit the model
'fitting' → often finding $\hat{\theta}$ that provides best explanation of the data
eg. Method of least Squares
↳ estimated parameters maximise probability of observing the data. ($\hat{\theta} = \text{argmax}_\theta \{p(x|\theta)\}$

Estimation Problems → Concerned by accuracy of θ
Prediction Problems → Concerned by accuracy of $p(x|\hat{\theta})$ - ie predicting new observations

What ML Problems Include

1) Dataset $\mathcal{D} = (X, y)$
↳ X - matrix of independent variables
y - vector of dependant variables
2) Model $f(x; \theta)$
↳ $f: x \to y$ of parameters θ
used to predict output from a vector of input variables
3) Cost function: $C(y, f(x, \theta))$
↳ Evaluates how well model performs on observations y
Model is fit by choosing θ s.t. Cost function minimised

How to obtain model:

1) Randomly divide Data Set into 'exclusive groups'
↳ Training Set: $\mathcal{D}_{train}$ (~90%), Test Set: $\mathcal{D}_{test}$ (~10%) → 'Cross-validation': mutually exclusive $\mathcal{D}_{test}$, $\mathcal{D}_{train}$
provides an unbiased estimate for the predictive performance of model
2) Minimized using cost function on training set
↳ $\hat{\theta} = \underset{\theta}{\arg\min} \{C(y_{train}, f(x_{train}; \theta))\}$
3) Analyse performance by evaluating Cost function on test data
↳ $C(y_{test}, f(x_{test}; \hat{\theta}))$

Errors:

Value minimised cost function on:
↳ training set → In Sample error, $E_{in} = C(y_{train}; f(x_{train}, \theta))$
test set → Out of Sample error, $E_{out} = C(y_{test}; f(x_{test}; \theta))$

Almost always true that:
↳ $E_{out} \geq E_{in}$

Often:
Model with Smallest $E_{out} \neq$ Model with Smallest E_{in} (ie. overfitting)

Often multiple candidate models to choose from
↳ Compared using E_{out} (minimised) → Test set is no longer independent of model decisions ('indirect tuning')
↳ 'optimism/selection bias' — performance likely (slightly) worse on new data.

Polynomial Regression – Out of Sample Performance

Start with:

Probabilistic Process:

$y_i = f(x_i) + n_i$ → Gaussian, uncorrelated noise: $\langle n_i \rangle = 0$ $\langle n_i n_j \rangle = \delta_{ij} \sigma^2$ σ noise strength

- f: fixed (known/unknown) funct
- $\langle n_i \rangle = 0$: mean/expectation value = 0
- $\langle n_i n_j \rangle$: covariance of noise; $\langle n_i n_j \rangle = 0$ when $i \neq j$ i.e. uncorrelated.

Case of polynomial regression:

- model class: $f_\alpha(x; \theta_\alpha)$

i.e Polynomials of order 1 (linear): $f_1(x; \underline{\theta}_1)$ → 2 parameters $\underline{\theta}_1$

Polynomials of order 3: $f_3(x; \underline{\theta}_3)$ → 4 parameters $\underline{\theta}_3$

Polynomials of order 10: $f_{10}(x; \underline{\theta}_{10})$ → 11 parameter $\underline{\theta}_{11}$

'More complese model "may" give us better predictive power but only if we have a large enough sample size to learn the model parameters'

Common difficulty → Predicting beyond training data

At Small Sample Sizes

→ Noise can be confused for fluctuations in genuine pattern

- Simple models: Cannot represent complicated patterns – forced to ignore noise; focuses on larger trends
- Complese models (many parameters): Can capture global trend + noise gen. pattern simultaneously; believes noise encodes real information } Overfitting

Avoiding overfitting:

1) Use less expressive models (fewer parameters)
2) Collect more data – likelihood noise appeared patterned reduced

} Bias variance trade off

When data restricted: Less expressive model

Statistical Learning Theory

Begin with: Unknown function, $y = f(x)$

Fixed hypothesis set; H

- (all functions willing to consider – usually dependent on intuition) defined on same domain as f

Observable data: $f(x)$ produces N sets of pairs (x_i, y_i); $i = 1, \ldots, N$

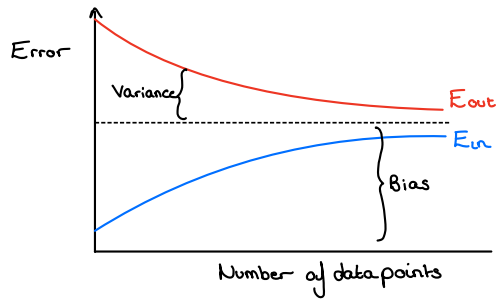
Goal: find function, $h \in H$, which best approximates $f(x)$

- Said to have 'learned' $f(x)$ if $h \approx f$
  - i.e. Will preform approximately well on new data as it did on training data ($E_{in} \approx E_{out}$)

Typical E_{in}, E_{out} as function of training set size

↳ true data is drawn from sufficiently complicated distribution

ie cannot learn function exactly



Infinite Data Limit

$N \to \infty$, $E_{in} \approx E_{out} \approx$ 'Bias' of model

Bias

↳ represents the best our model could if in $N = \infty$

feature of model class used (constant for given complexity)

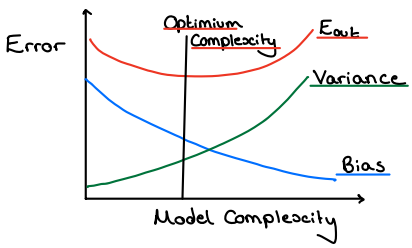
decreases with model class complexity

$E_{out} - E_{in}$

↳ Difference between fitting and predicting

In finite data limit

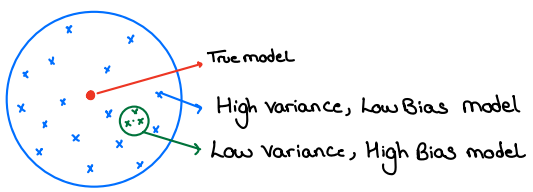
↳ to get best predictive power: minimise E_{out} rather than bias



Shown above $E_{out} = \text{Bias} + \text{Variance}$

Bias : Limit of what could be achieved with infinite data

Variance: Errors from training with sampling noise on finite data set



'At some point model becomes too complex for the amount of training data — high variance'

Bias - Variance Trade off

Bias-Variance Decomposition

Consider data set $\mathcal{D}(\underline{x}, \underline{y})$ consisting of N pairs of independant and dependant variables.

True data from noisy model: $y = f(\underline{x}) + \varepsilon$

↳ Normally dist: Mean = 0, Std = σ_ε

Cost Function (ie least squares regression)

↳ $C(\underline{y}, f(\underline{x}, \underline{\theta})) = \sum_i (y_i - f(\underline{x}_i; \underline{\theta}))^2$

Estimates of parameters: $\hat{\underline{\theta}}_{\mathcal{D}} = \underset{\theta}{\text{argmin}}\, C(\underline{y}; f(\underline{x}; \underline{\theta}))$

↳ function of dataset

Different error for each Dataset $D_j = (\underline{y}_j, \underline{x})$

↳ Error: $C(y_i, f(\underline{x}_i; \hat{\underline{\theta}}_{\mathcal{D}_j}))$

Over N samples from true data distribution:

↳ Expectation Value over all $\mathcal{D}_j$: $\mathbb{E}_{\mathcal{D}}$

Expectation Value over different noise, ε: $\mathbb{E}_\varepsilon$

} Expected Generalisation Error: $\mathbb{E}_{\mathcal{D},\varepsilon}$

↳ Over both samples, $\mathcal{D}$ and noise, ε

Expectation Generalisation Error

$$\mathbb{E}_{\mathcal{D},\varepsilon}\left[C\left(\underline{y}, \underline{f}(\underline{x}, \underline{\theta}_{\mathcal{D}})\right)\right]$$

$$= \mathbb{E}_{\mathcal{D},\varepsilon}\left[\sum_i \left(y_i - f(\underline{x}_i; \hat{\underline{\theta}}_{\mathcal{D}})\right)^2\right]$$ → [least squares error] across all $\mathcal{D}$, ε

$$= \mathbb{E}_{\mathcal{D},\varepsilon}\left[\sum_i \left(y_i - f(\underline{x}_i) + f(\underline{x}_i) - f(\underline{x}_i; \hat{\underline{\theta}}_{\mathcal{D}})\right)^2\right]$$

$$= \sum_i \mathbb{E}_{\varepsilon}\left[(y_i - f(x_i))^2\right] + \mathbb{E}_{\mathcal{D},\varepsilon}\left[(f(\underline{x}_i) - f(\underline{x}_i; \hat{\underline{\theta}}_{\mathcal{D}}))^2\right] + 2\mathbb{E}_{\varepsilon}\left[y_i - f(\underline{x}_i)\right]\mathbb{E}_{\mathcal{D}}\left[f(\underline{x}_i) - f(\underline{x}_i; \hat{\theta}_{\mathcal{D}})\right]$$

- $\sum_i \mathbb{E}_{\varepsilon}\left[(y_i - f(x_i))^2\right]$: Difference between observed value y_i and true function. As $y_i = f(x_i) + \varepsilon$, $y_i - f(x_i) = \varepsilon$
- $\mathbb{E}_{\mathcal{D},\varepsilon}\left[(f(\underline{x}_i) - f(\underline{x}_i; \hat{\underline{\theta}}_{\mathcal{D}}))^2\right]$: Difference between modelled and true function: model error. Also indep of ε: $\mathbb{E}_{\mathcal{D},\varepsilon} \to \mathbb{E}_{\mathcal{D}}$
- $2\mathbb{E}_{\varepsilon}\left[y_i - f(\underline{x}_i)\right]\mathbb{E}_{\mathcal{D}}\left[f(\underline{x}_i) - f(\underline{x}_i; \hat{\theta}_{\mathcal{D}})\right]$: ε → Error has mean zero; $\mathbb{E}_{\varepsilon}[\varepsilon] = 0$ ∴ Cross term vanishes.

$$= \sum_i \underbrace{\mathbb{E}_{\varepsilon}[\varepsilon^2]}_{\sigma_\varepsilon^2} + \mathbb{E}_{\mathcal{D}}\left[(f(\underline{x}_i) - f(\underline{x}_i; \hat{\underline{\theta}}_{\mathcal{D}}))^2\right] = \sum_i \sigma_\varepsilon^2 + \mathbb{E}_{\mathcal{D}}\left[(f(\underline{x}_i) - f(\underline{x}_i; \hat{\underline{\theta}}_{\mathcal{D}}))^2\right]$$

Further expanding:

$$\mathbb{E}_{\mathcal{D}}\left[(f(\underline{x}_i) - f(\underline{x}_i; \hat{\underline{\theta}}_{\mathcal{D}}))^2\right]$$

± Expected prediction $E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})]$

$$= \mathbb{E}_{\mathcal{D}}\left[\left\{f(\underline{x}_i) - E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})] + E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})] - f(x_i; \hat{\theta}_{\mathcal{D}})\right\}^2\right]$$

$$= \mathbb{E}_{\mathcal{D}}\left[\left\{f(x_i) - E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})]\right\}^2\right] + E_{\mathcal{D}}\left[\left\{E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})] - f(x_i; \hat{\theta}_{\mathcal{D}})\right\}^2\right] + 2\mathbb{E}_{\mathcal{D}}\left[\left\{f(x_i) - E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})]\right\}\left\{E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})] - f(x_i; \hat{\theta}_{\mathcal{D}})\right\}\right]$$

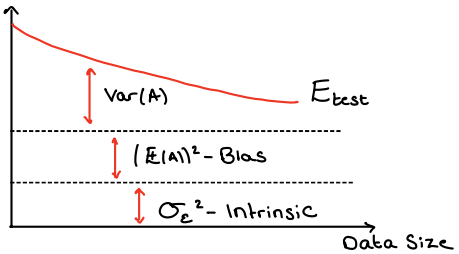
- $\left\{f(x_i) - E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})]\right\}^2$: Difference between true function and the average predicted over $\mathcal{D}$ → Independent of $\mathcal{D}$ ($\mathbb{E}[\,]$ irrelevant)
- Deviation of true from average predicted: Independent of $\mathcal{D}$ (Constant)
- Deviation of models predicted from its mean: $E[\text{Deviation from mean}] = 0$

$$= \left\{f(x_i) - E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})]\right\}^2 + E_{\mathcal{D}}\left[\left\{E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})] - f(x_i; \hat{\theta}_{\mathcal{D}})\right\}^2\right]$$

Overall: $\mathbb{E} = \sum_i \sigma_\varepsilon^2 + \left\{f(x_i) - E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})]\right\}^2 + E_{\mathcal{D}}\left[\left\{E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})] - f(x_i; \hat{\theta}_{\mathcal{D}})\right\}^2\right]$

- **Bias²:** $\sum_i \left\{f(x_i) - E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})]\right\}^2$ → Measures deviation of expectation value of our estimator from the true value
- **Var:** $E_{\mathcal{D}}\left[\left\{E_{\mathcal{D}}[f(x_i; \hat{\theta}_{\mathcal{D}})] - f(x_i; \hat{\theta}_{\mathcal{D}})\right\}^2\right]$ → Measures how much estimator fluctuates due to finite-sample effects
- **Noise:** $\sum_i \sigma_\varepsilon^2$

$$\mathbb{E} = \text{Bias}^2 + \text{Var} + \text{Noise}$$



Gradient Descent and its Generalizations

All ML Start with:

Dataset, X model $g(\theta)$ - function of parameters θ Cost function $C(X, g(\theta))$

Fitting: θ that minimise Cost function

Method: Gradient Descent

↳ Iteratively adjust θ

In direction which gradient Cost function: Large + Negative

Ensure approach local minimum $C(X, g(\theta))$

Complications: Cost function often - Complese
- Non Convese
- dont actually know function to minimise
- high dimensional Space
- many local minima

Gradient Descent

→ Wish to minimise '$E(\theta) = C(X, g(\theta))$'

Initialise parameters – θ_0

Iteratively update – $V_t = \eta_t \nabla_\theta E(\theta_t)$

↳ Learning rate: η_t

Controls step size in direction of Gradient

Gradient of $E(\theta)$ wrt θ $\nabla_\theta E(\theta_t)$

$\theta_{t+1} = \theta_t - V_t$

Side note ($E(\theta)$ - 'Energy Function')

Written as Sum over n data points:

$$E(\theta) = \sum_{i=1}^{n} e_i(x_i, \theta)$$

Where for Linear Regression, e_i – Mean Square error

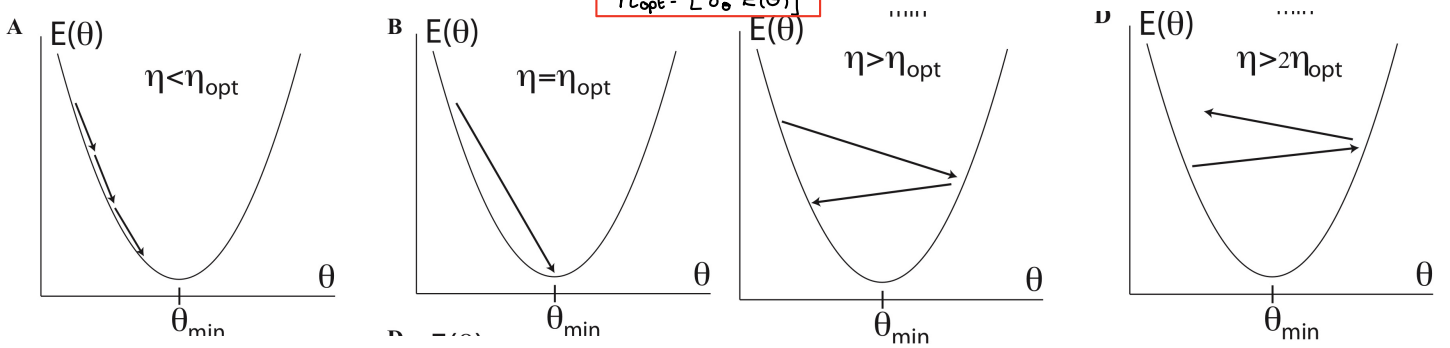
Logistic Regression, e_i - Cross entropy

Regimes for learning rate

η_{opt} → optimal learning rate to take us to minimum in 1 step

From Newton's Method: Minimum taylor expansion

$$\eta_{opt} = [\partial_\theta^2 E(\theta)]^{-1}$$



GD Coverges to min in multiple Steps

GD Converges to min in 1 Step

GD oscilattes around min Slowly Converging

GD oscillates while diverging from min

Limitations (Simple Gradient Descent)

→ GD finds local minima of cost function
  ↳ ML often many local minima
  Solution: Introduce Stochasticity

→ Gradients computationally expensive to calculate for large datasets (ie sum over n data points)
  ↳ Solution: Calculate gradients using small subsets 'mini-batches'. → This also adds Stochasticity !!

→ GD treats all directions in parameter space uniformly
GD is very sensitive to choice of learning rates.
  ↳ Solution: adaptively choose the learning rates

→ GD sensitive to initial conditions
  ↳ Can take exponential time to escape saddle points which are common in high dimensional space

Newton's Method

Choose Step V which minimises 2nd order taylor expansion of Energy/cost function: $E(\underline{\theta}+\underline{V}) \approx E(\underline{\theta}) + \nabla_\theta E(\underline{\theta})\, V + \frac{1}{2} V^T H(\theta) V$
  ↳ Hessian matrix of second derivatives

Differentiating equation wrt V
  ↳ to find minimium (approx)
  Can Calculate V_{opt} → optimal step to take us to minimium in 1 step
  Note with V_{opt}: $\nabla_\theta E(\theta + V_{opt}) = 0$

Get: $\nabla_\theta E(\underline{\theta}) + H(\underline{\theta}) V_{opt} = 0 \rightarrow \boxed{V_{opt} = -H^{-1} \nabla_\theta E}$

Achieve Iteration Steps $\boxed{V_t = H^{-1}(\theta_t)\, \nabla_\theta E(\underline{\theta}_t)}$

$$\boxed{\theta_{t+1} = \theta_t - V_t}$$

Disadvantages/Impracticalities Newton Methods
  ↳ Hessian is an extremely expensive numerical computation
  2nd order derivative information for Hessian - n^2 entries (n-params)

Important lessons from Newton Method
  ↳ Adapt learning rate of different parameters depending on Hessian
    ↳ Hessian shows curvatures of surface
      Automatically: Larger steps in flat directions (small curvature)
      Smaller steps in steep directions (large curvature)

Updated Gradient Descent Methods

Stochastic Gradient Descent (with minibatches)
  ↳ Stochatisity achieved by:
    Only using subset of data (minibatch) to estimate gradient (random mini sets of data)
    Minibatch Sizes 10's - 100's datapoints
    n, total data points
    m, Size minibatches
    } n/m minibatches, B_k $k = 1, \ldots, n/m$

At each gradient descent step:

Approx gradient using Single mini batch, B_k

$$\nabla_\theta E(\underline{\theta}) = \sum_{i=1}^{n} \nabla_\theta e_i(\underline{x}_i, \underline{\theta})$$

$$\downarrow$$

$$\nabla_\theta^{MB}(\underline{\theta}) = \sum_{i \in B_k} \nabla_\theta e_i(\underline{x}_i, \underline{\theta})$$

We use each minibatch gradient approx to update parameters $\underline{\theta}$ at every step k

Next update $\longrightarrow$ Next minibatch

Once cycled over all n/m minibatches $\longrightarrow$ 1 epoch

Normally reshuffles data into 'fresh' minibatches after each epoch.

$$V_t = \eta_t \nabla_\theta E^{MB}(\theta)$$

$$\downarrow$$

$$\theta_{t+1} = \theta_t - V_t$$

Advantages: Decreases Chance of getting Stuck in local minimum

Rapidly Speeds up Calculation time.

Adding Momentumn (Inertia term) to Gradient Descent (GDM)

↳ Momentumn Serves as memory of the direction of movement in parameter Space

Implemented using:

$$V_t = \gamma V_{t-1} + \eta_t \nabla_\theta E(\theta_t) \longrightarrow \theta_{t+1} = \theta_t - V_t$$

↳ Memory term (γV_{t-1})

Momentumn parameter, γ

$0 \leq \gamma \leq 1$

Alternatively Written: (all in one) $\Delta\theta_{t+1} = \gamma \Delta\theta_t - \eta_t \nabla_\theta E(\theta_t)$

V_t becomes 'running average of recently encountered gradients'

↳ $(1-\gamma)^{-1}$: Characteristic time Scale for memory

Similarities with: Movement of partical with mass, m moving through viscous medium (ie viscous damping)

Advantages: GD algorithms gains Speed in direction with persistent but Small gradients

Surpresses oscillations (Speed) in high Curvature directions

Even further: Nesterov Accelerated Gradient

Calculates the gradient at excpected value of parameters given current Momentumn:

$\nabla_\theta E(\theta_t) \longrightarrow \nabla_\theta E(\theta_t + \gamma V_{t-1})$

Thus: $V_t = \gamma V_{t-1} + \eta_t \nabla_\theta E(\theta_t + \gamma V_{t-1})$

$$\downarrow$$

$\theta_{t+1} = \theta_t - V_t$ NAG Update Rule

Adding 2nd moment of gradient

With or Without Momentumn

$\longrightarrow$ must Specify Specify 'Schedule' for tuning η as function of time.

Second order methods → Calculate / approximate Hessian

Normalise learning rate by curvature

Large steps → Shallow Small steps → Steep

But computationally expensive

RMS Prop Algorithm

↳ Keep track of second moment, $S_t = \mathbb{E}[g_t^2]$ (as well as first)

$$g_t = \nabla_\theta E(\theta)$$

$$S_t = \beta S_{t-1} + (1-\beta)\, g_t^2$$

$$\theta_{t+1} = \theta_t - \eta_t \frac{g_t}{\sqrt{S_t + \epsilon}}$$

β - Controls averaging time for second momentumn.

Similiarities

effective step size of parameters dependent on gradient magnitude squared.

Learning rates for RMSprop and Adam can be set significantly higher due to adapt. Step sizes.

ADAM Optimiser

↳ Keeps track of 1st, 2nd moment of gradient ($m_t = \mathbb{E}[g_t]$; $S_t = \mathbb{E}[g_t^2]$)

→ Performs an additional bias correction

↳ Accounts for fact first two moments of gradient are calc with 'running average'

$$g_t = \nabla_\theta E(\theta)$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \qquad \hat{m}_t = \frac{m_t}{1-(\beta_1)^t}$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2) g_t^2 \qquad \hat{S}_t = \frac{S_t}{1-(\beta_2)^t}$$

$$\theta_{t+1} = \theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{S}_t} + \epsilon}$$

β_1, β_2 - Set memory lifetime of first and second moment

ADAM Benefits

1) adapting step size to cut off larg gradient directions → prevent oscillations/ divergences.

2) Gradients measured in terms of natural length scale. Std σ_t

Written in terms of variance ($\sigma_t^2 = \hat{S}_t - (\hat{m}_t)^2$)

$$\Delta\theta_{t+1} = -\eta_t \frac{\hat{m}_t}{\sqrt{\sigma_t^2 + \hat{m}_t^2 + \epsilon}}$$

Limiting Cases

↳ Gradient estimates consistent (ie small variance $\sigma_t^2 << m_t^2$)

$\Delta\theta_{t+1} \longrightarrow -\eta_t$ (assuming $\hat{m}_t << \epsilon$)

Cuts of large persistent gradients and limits max step size in steep directions

Gradients wildly fluctuating (ie large variance $\sigma^2 >> m_t^2$)

$$\Delta\theta_{t+1} \longrightarrow -\frac{\eta_t \hat{m}_t}{\sigma_t}$$

Adapt learning rate such that preportional to signal-to-noise ratio (SNR) m_t/σ_t

Gradient Descent - Practical tips

→ Randomly Shuffle data when making minibatches

↳ Otherwise fits correleations to order of data presented.

→ Transform your inputs

↳ Standardise data by subtracting mean and normalising variance of input variables

→ Moniter out of Sample performance

↳ As E_{out} (or validation set) increases

Overfitting → Terminate learning process 'earlyStopping' improves performance

Bayesian Inference

↳ Instead of finding 'single' best parameter values, deals with 'posterior' distribution of parameters given data.

↳ Treats parameters as random variables with distributions.

Gradient Descent (frequentist machine learning)

Bayesian Inference (bayesian machine learning)

Bayes Rule

$$p(\underline{\theta}|\underline{x}) = \frac{p(\underline{x}|\underline{\theta})\,p(\underline{\theta})}{\int d\underline{\theta}'\, p(\underline{x}|\underline{\theta}')p(\underline{\theta}')}$$

Likelihood function: $p(\underline{x}|\underline{\theta})$

↳ Probability observing a dataset $\underline{x}$ for given value unknown parameters $\underline{\theta}$

Considered function $\underline{\theta}$, with data $\underline{x}$ held constant.

Prior Distribution, $p(\underline{\theta})$

↳ Any prior knowledge of parameters

Posterior Distribution $p(\underline{\theta}|\underline{x})$

↳ Knowledge of unknown parameter $\underline{\theta}$ after observing $\underline{x}$

Maximum Likelihood Estimator:

Selection of parameters that maximise (log of) likelihood function

$$\hat{\underline{\theta}} = \underset{\underline{\theta}}{\text{argmax}} \{\log p(\underline{x}|\underline{\theta})\}$$

→ Applies to frequentist and Bayesian

ie Maximise Probability of seeing observed data given generative model.

Two Classes Prior

→ **Uninformative prior:**

↳ no specialist knowledge about $\underline{\theta}$ before data

Informative prior:

↳ reflects level of knowledge of $\underline{\theta}$

Tends to: decrease variance of posterior dist, increase bias } Usefulness: Decrease in variance > increase in Bias

In High-dimensional problems — many parameters zero (or close to)

Gaussian Prior

$$p(\underline{\theta}|\lambda) = \prod_j \sqrt{\frac{\lambda}{2\pi}}\, e^{-\lambda\theta_j^2}$$

Laplace Prior

$$p(\underline{\theta}|\lambda) = \prod_j \frac{\lambda}{2}\, e^{-\lambda|\theta_j|}$$

→ 'many parameters small'

Choosing $\underline{\theta}$

Either: **Bayes estimate** — Posterior mean: $\langle\underline{\theta}\rangle = \int d\underline{\theta}\, \underline{\theta}\, p(\underline{\theta}|\underline{x})$

Minimises mean square error

gives fuller understanding of uncertainty

or **MAP (maximum-a-posteriori) Estimate** — Posterior mode: $\hat{\underline{\theta}}_{MAP} = \underset{\underline{\theta}}{\text{argmax}}\, p(\underline{\theta}|\underline{x})$

All methods for choosing $\underline{\theta}$ of model

→ GD → MAP Estimate $\underline{\theta}_{MAP}$

MLE Posterior Mean $\langle\underline{\theta}\rangle$

Linear Regression — n Sample - each with p features

Dataset of n samples: $\mathcal{D} = \{(y_i, \underline{x}^{(i)})\}_{i=1}^{n}$ Where $\underline{x}^{(i)}$ – i^{th} observation vector

→ Assume each sample p features: $\underline{x}^i \in \mathbb{R}^p$

y_i - Corresponding scalar response

True function/Model: $y_i = f(\underline{x}^{(i)}; \underline{w}_{true}) + \epsilon_i$ where $\underline{w}_{true} \in \mathbb{R}^p$ - true parameter vector

ϵ_i – iid noise with mean zero and finite variance.

Design matrix $X \in \mathbb{R}^{n \times p}$ $n \times p$ matrix where each row is an observation and the columns the measured features

Rows: $X_{i,:} = \underline{x}^{(i)} \in \mathbb{R}^p$ $i = 1, \dots, n$

Columns: $X_{:,j} \in \mathbb{R}^n$ $j = 1, \dots, p$

Linear Regression:

Assume $y_i = f(\underline{x}^{(i)}, \underline{w}_{true}) + \epsilon_i$

$y_i \approx \underline{w}_{true}^T \underline{x}^{(i)} + \epsilon_i$ → Vector multiplication - linear

Wish to find function g that best approximates f

Notation: L^p norm of a vector

$p \in \mathbb{R}$ and $p \geq 1$ $\underline{x} = (x_1, \dots, x_d) \in \mathbb{R}^d$

$\|\underline{x}\|_p = (|x_1|^p + \dots + |x_d|^p)^{1/p}$

Least Squares Regression

'Minimisation of the L_2 norm of the difference between the response y_i and predictor $g(\underline{x}^{(i)}; \underline{w}) = \underline{w}^T \underline{x}^{(i)}$:' } finding $\underline{w}$ to minimise L_2 error

$$\hat{w}_{LS} = \underset{w \in \mathbb{R}^p}{\operatorname{argmin}} \|X\underline{w} - \underline{y}\|_2^2 = \underset{w \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{i=1}^{n} (\underline{w}^T \underline{x}^{(i)} - y_i)^2$$

Note: Predictor $g(\underline{x}^i; w) = \underline{w}^T \underline{x}^{(i)}$ $\hat{w}_{LS}$ → defines hyperplane in $\mathbb{R}^p$

Minimising LSE ≈ Minimising sum projections/residuals to hyperplane

Through differentiation: $\frac{\partial}{\partial w} (x_{ij} w_j - y_i)(x_{ik} w_k - y_i) = 0$

→ **Einstein notation:** Summing over repeated indices

$= 2(x_{ij} w_j - y_i) x_{im}$ i - data points j, k - features

$$\hat{w}_{LS} = (X^T X)^{-1} X^T y$$

→ Assuming $X^T X$ is invertible (often case $n >> p$)

→ If $X^T X$ is singular ($p > n$ - more features than samples)

→ Infinitely many solutions to Least Squares problem.

Must introduce 'regularisation'

→ Constraints that ensure unique stable solution

∴ **Best fit of data X** $\hat{y} = X \hat{w}_{LS} = P_X y$

Where $P_X = X(X^T X)^{-1} X^T$ → Projection matrix: acts on y and projects into column space X

Provided we can obtain Soln $\hat{w}_{LS}$

$$\bar{E}_{in} = \sigma^2\left(1 - \frac{p}{n}\right) \qquad \bar{E}_{out} = \sigma^2\left(1 + \frac{p}{n}\right)$$

Average generalisation error:

$$|\bar{E}_{in} - \bar{E}_{out}| = 2\sigma^2\frac{p}{n}$$

→ If $p \gg n$ – High Dimensionality or $n \approx p$

Generalisation error is large

↳ Introduce regularization

Ridge Regression (penalized regression problem)

↳ L_2 penalty: regularizer – 'L_2 norm of the parameter vector'

$$\hat{w}_{Ridge}(\lambda) = \underset{w \in \mathbb{R}^p}{\operatorname{argmin}} \left(\|Xw - y\|_2^2 + \lambda\|w\|_2^2\right) \quad \text{where } \lambda > 0$$

$\simeq$ Equivalence

$$\hat{w}_{Ridge} = \underset{w \in \mathbb{R}, \|w\|_2 \leq t}{\operatorname{argmin}} \|Xw - y\|^2 \quad t > 0$$

↳ Adding regularization term: $\lambda\|w\|_2^2$

↳ Constraining magnitude of parameter vector learned from data

Solving by differentiation:

$$\frac{\partial}{\partial w}\left(\|Xw - y\|_2^2 + \lambda\|w\|_2^2\right) = 2(x_{ij}w_j - y_i)x_{im} + 2\lambda w_m = 0$$

$$-y_i x_{im} + \lambda w_m + x_{ij}w_j x_{im} = 0$$

$$(X^TX + \lambda I_{p\times p})w - X^Ty = 0$$

Assuming X^TX invertible

$$\hat{w}_{Ridge} = (X^TX + \lambda I_{p\times p})^{-1}X^Ty$$

→ If X orthogonal: further simplified

$$\hat{w}_{Ridge}(\lambda) = \frac{\hat{w}_{LS}}{1+\lambda}$$

↳ Just least squared estimate scaled by $(1+\lambda)^{-1}$

Single Value Decomposition

$$X = UDV^T$$

$U \in \mathbb{R}^{n\times p}$ → Columns Span Column Space of X

$V \in \mathbb{R}^{p\times p}$ → rows Span row Space of X

$D \in \mathbb{R}^{p\times p}$ → $\text{diag}(d_1, \ldots, d_p)$ – d_i Singular values of X

$$\hat{w}_{ridge} = (X^TX + \lambda I_{p\times p})^{-1}X^Ty \xrightarrow{X = UDV^T} \hat{w}_{ridge} = V(D^2 + \lambda I)^{-1}DU^Ty$$

Ridge predictor: $\hat{y}_{ridge} = X\hat{w}_{ridge}$

$$= \sum_{j=1}^{p} U_{:,j}\frac{d_j^2}{d_j^2 + \lambda}U_{:,j}^Ty \leq UU^Ty \equiv \hat{y}_{LS}$$

$$\therefore \hat{y}_{Ridge} \leq \hat{y}_{LS}$$

↳ Ridge further shrinks each basis component j by factor $\frac{d_j^2}{(d_j^2 + \lambda)}$.

Ridge:

discourages large parameters by shrinking them

↳ reduces overfitting and stabalises Soln.

Lasso Regression

↳ L_1 penalty: regularizer - Least absolute Shrinkage and Selector operator

$$\hat{w}_{Lasso}(\lambda) = \underset{w \in \mathbb{R}^p}{\operatorname{argmin}} \|Xw - y\|_2^2 + \lambda \|w\|_1 \quad \approx \quad \hat{w}_{Lasso}(t) = \underset{w \in \mathbb{R}^p : \|w\|_1 \leq t}{\operatorname{argmin}} \|Xw - y\|_2^2$$

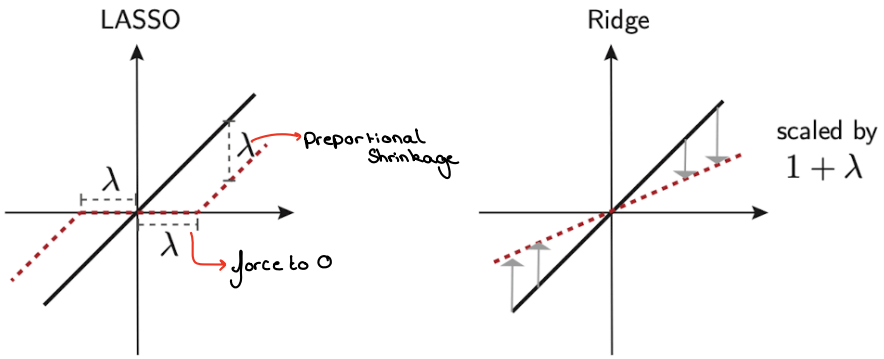
Equivelence

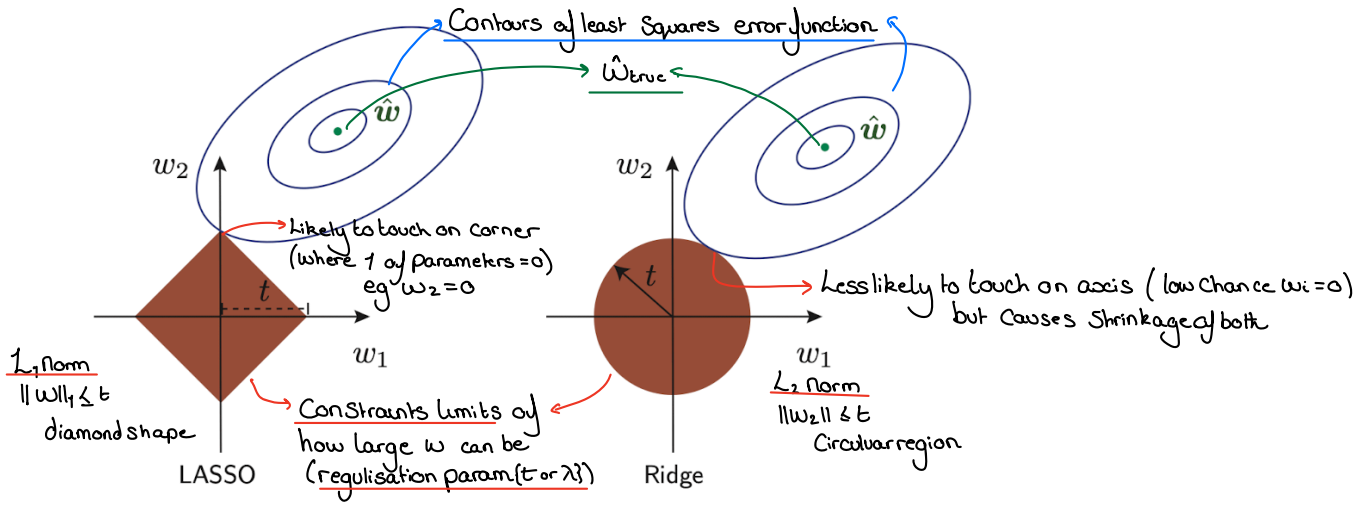
Cannot take $\partial/\partial w$ as L_1 regularizer is not everywhere differentiable

For X orthogonal $\hat{w}_j^{lasso}(\lambda) = \text{Sign}(\hat{w}_j^{LS})(|w_j^{LS}| - \lambda)_+$

↳ denotes +ve part of (...)

↳ Lasso: 'Soft-thresholding' - Small Coeff → 0, large Coeff → Shrunk preportionally
Shrinks many component $\hat{w}_{Lasso}$ to 0
Effectively removes less important features.





Linear Regression for Hamiltonian of Ising Model

Ensemble of random Spin Configurations (10)

$$H = -J \sum_{j=1}^{L} S_j S_{j+1}$$

J - nearest neighbour Spin Interaction

$S_j \in \{\pm 1\}$ - Spin variable (assume generated with $J = 1$)

Handed Dataset $\hat{D} = \left(\{S_j\}_{j=1}^{L}, E_j\right)$

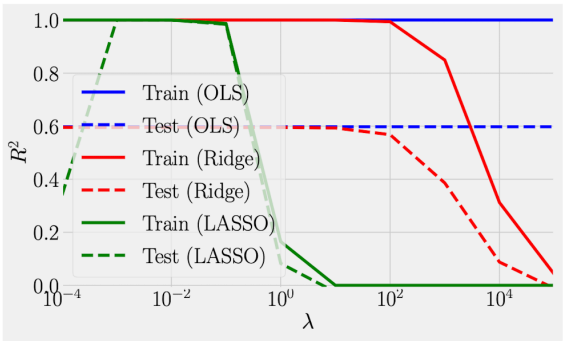
Model Class: 'Spin model with pairwise interactions between every pair of variable': $H_{model}[S^i] = -\sum_{j=1}^{L}\sum_{k=1}^{L} J_{j,k} S_j^i S_k^i$

Goal: Determing $J_{i,k}$ matrice from linear regression on $\hat{D}$

Linear Regression form: $H_{model}[S^i] = \underline{X^i} \cdot \underline{J}$

↳ all two body interactions $\{S_j^i S_k^i\}_{j,k=1}^{L}$

Performance Measured: $R^2 = 1 - \frac{\sum_{i=1}^{n} |y_i^{true} - y_i^{pred}|^2}{\sum_{i=1}^{n} |y_i^{true} - \frac{1}{n}\sum_{i=1}^{n} y_i^{pred}|^2}$



↳ Tuning λ - hyper parameter tuning

Bayesian formulation of Linear Regression

Data $\hat{D}$ used to fit regression model

↳ Generated through: $y = \underline{x}^T \underline{w} + \epsilon$

↳ ϵ, Gaussian noise → mean - 0

Variance - σ^2

Linear regression model in Bayesian Framework

$p(y|\underline{x}, \underline{\theta}) = \mathcal{N}(y|\mu(\underline{x}), \sigma^2(\underline{x}))$

↳ $\underline{\theta} = (w, \sigma^2)$

Maximum Likelihood Estimation (MLE)

↳ Defined by maximising log-likelihood wrt $\underline{\theta}$

$\hat{\theta} = \arg\max_{\theta} \log p(\hat{D}|\theta)$ → Assuming Samples are iids → $l(\theta) = \log p(D|\theta) = \sum_{i}^{n} \log(p(y_i|x_j^{(i)}\theta)$

$l(\theta) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - w^T x^{(i)})^2 - \frac{n}{2}\log(2\pi\sigma^2)$

$l(\theta) = -\frac{1}{2\sigma^2}||Xw - y||_2^2 + \text{Const}$ → performing least squares is equivelent to maximising log likelihood.

MLE ↔ Least Squares

Maximum a posteriori probability (MAP) estimate

regularized linear regression

$p(\underline{\theta}|D) \propto p(\hat{D}|\underline{\theta})p(\underline{\theta})$

Maximise log posterior

$\hat{\theta}_{MAP} \equiv \arg\max_{\theta}(\log p(\hat{D}|\theta) + \log p(\theta))$

Example: Gaussian prior (zero mean, variance τ^2)

↳ $p(w) = \prod_j \mathcal{N}(w_j|0, \tau^2)$

$\hat{\theta}_{MAP} = \arg\max_{\theta}\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - w^T x^{(i)})^2 - \frac{1}{2\tau^2}\sum_{j=1}^{n} w_j^2\right]$ $\hat{\theta}_{MAP} = \arg\max_{\theta}\left[-\frac{1}{2\sigma^2}||Xw - y||_2^2 - \frac{1}{2\tau^2}||w||_2^2\right]$

Basis function Expansion

Linear regression used to model non linear relationship between input and responce

Replace Input: $\underline{x} \longrightarrow$ non-linear function $\phi(\underline{x})$

↳ Preserve linearity as a function of w

$\phi^T(\underline{x})\,\underline{w}$

Logistic Regression

↳ deals with binary, dichotomous outcomes

↳ 'distinct mutually exclusive catagories'

Previously: Linear Regression

↳ 'Continuous' outputs - y_i from independent variables $\underline{x}_i$

Consider case where dependent variables $y_i \in \mathbb{Z}$

↳ discrete, take values $m = 0, \ldots, M-1$

Goal: Predict output Classes from the design matrix $X \in \mathbb{R}^{n \times p}$ (n Samples, p features)

Converting discrete variables

Perceptron

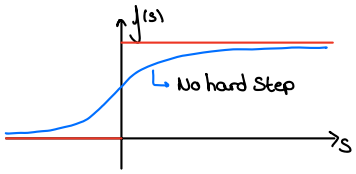
⟶ maps linear regressor to $\{0,1\}$

$$\sigma(s_i) = \text{Sign}(s) = \begin{cases} 1 & s_i \geq 0 \\ 0 & s_i < 0 \end{cases}$$

Hard classification

Set backs: do not provide confidence level (probability)

do not account for margins of Seperation



Logistic/Sigmoid Function

$$\sigma(s) = \frac{1}{1+e^{-s}}$$

Soft classification

↳ Property of note: $1 - \sigma(s) = \sigma(-s)$

Probability $\underline{x}_i$ belongs to Catagory $y_i = \{0,1\}$

$$P(y_i = 1 \mid \underline{x}_i, \underline{\theta}) = \frac{1}{1+e^{-x_i^T\theta}}$$

$$P(y_i = 0 \mid \underline{x}_i, \underline{\theta}) = 1 - P(y_i = 1 \mid x_i, \theta) = \frac{e^{-x_i^T\theta}}{1+e^{-x_i^T\theta}}$$

⟶ Similiar to two state System in Statistical physics

Cost function using MLE $(C(w) = -l(w))$

Consider Dataset $D = \{(y_i, \underline{x}_i\}$ with binary labels $y_i \in \{0,1\}$

Likelihood of observing data under our model:

$$P(D \mid w) = \prod_{i=1}^{n} [\sigma(\underline{x}_i^T w)]^{y_i} [1 - \sigma(x_i^T w)]^{1-y_i}$$

⟶ Binomial probability function where p(success) = σ

σ - Sigmoid function - outputs a probability $\{0,1\}$

$$l(\underline{w}) = \sum_{i=1}^{n} y_i \log \sigma(x_i^T w) + (1-y_i) \log [1 - \sigma(\underline{x}_i^T \underline{w}]$$

$$\text{MLE: } \hat{\underline{w}} = \arg\max_{\theta} \sum_{i=1}^{n} y_i \log \sigma(x_i^T w) + (1-y_i) \log [1 - \sigma(x_i^T w)]$$

Cross Entropy - Cost function of Logistic Regression

$$C(w) = -l(-w) = \sum_{i=1}^{n} -y_i \log \sigma(x_i^T w) - (1-y_i)\log[1-\sigma(x_i^T w)]$$

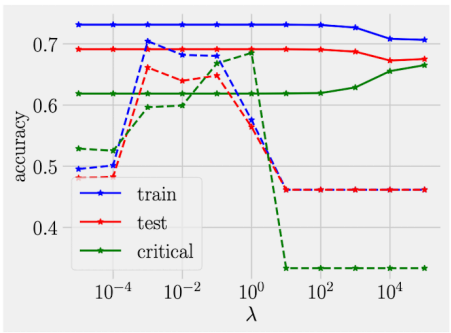
→ Similiar to Regularization of linear regression often Suplimented with extra terms
↳ L_1 and L_2 Regularisation

↳ Convex function of weights
Any local minimizer is a global minimizer.

$$0 = \nabla C(w) = \sum_{i=1}^{n} [\sigma(x_i^T w) - y_i] x_i$$

↳ Solved numerically no closed form.

Used: $\partial_z \sigma(s) = \sigma(s)[1-\sigma(s)]$



Sweet Spot for SGD regularization Strength λ - results in optimal performance of logistic regressor.

Example of Binary Classification

2D Ising model

Hamiltonian: $H = -J \sum_{\langle ij \rangle} S_i S_j$ $\quad S_j \in \{\pm 1\}$

J, Interaction Energy Scale

↳ i,j run over all nearest neighbours of a 2D Square lattice

Goal: Determine whether a state is ordered or disordered.

Softmax Regression

Generalises Logistic regression to mult-class classification

Label $y_i \in \mathbb{Z}_2^M$ binary string of length M

↳ $y_i = (0,\dots,0,\overset{0}{1},0,\dots 0)$ (1 at position m, $M-1$ entries)

↳ x_i belongs to class —

Probability of x_i being in class m

$$P(y_{im'} = 1 \mid x_i, \{w_k\}_{k=0}^{M-1}) = \frac{e^{-x_i^T w_{m'}}}{\sum_{m=0}^{M-1} e^{-x_i^T w_m}}$$

↳ $y_{im'} \equiv [y_i]_{m'}$

Likelihood of M-Class classifier

$$P(D \mid \{w_k\}_{k=0}^{M-1}) = \prod_{i=1}^{n} \prod_{m=0}^{M-1} [P(y_{im}=1 \mid x_i, w_m)]^{y_{im}} [1 - P(y_{im}=1 \mid x_i, w_m)]^{1-y_{im}}$$

Cost function

$$C(w) = -\sum_{i=1}^{n} \sum_{m=0}^{M-1} y_{im} \log(y_{im}=1 \mid x_i, w_m) + (1-y_{im}) \log(1 - P(y_{im}=1 \mid x_i, w_m))$$

Capacity of a Single Neuron

K inputs (parameters of model)

Data set $\mathcal{D}_N$ is labelling of N points ⟶ Ideally 2^N ways of Classifying

> **'Memory'** - Capacity to Correctly Classify (memorise) distinct patterns of binary labels
>
> ↳ No. of unique ways Neuron can assign 0's, 1's to a given set of points in space

Assume: All inputs $\{\underline{x}_n\}$ are in general position

General Position

↳ For a Set of $\{\underline{x}_n\}$ points in K-dimensional Space, are said to be in 'General Position' if:

↳ Any Subset of Size $\leq K$ is linearly independent

No K+1 of them lie on a (K-1) dimensional plane

ie K=3 dimensions ⟶ No 3 points Colinear

No 4 points are Coplanar

If Sigmoid function used instead

↳ Difference: Smooth output (Continuous [0,1])

Similiarity: Decision Boundary remains linear.

Linear Threshold Function

$$y = f\left(\sum_{k=1}^{K} w_k x_k\right) \quad \text{where} \quad f(a) = \begin{cases} 1 & a > 0 \\ 0 & a \leq k \end{cases}$$

⟶ **Single neuron/perceptron**

↳ distinct binary classification based on position of each point

Linear decision boundary (ie straight line in 2D Space)

↳ Point falls 1 side of line: output-1

Point falls other side of line: output-0

Thus limited capacity for the number of labelings it can Correctly Classify.

No. of threshold functions, $T(N,K)$

↳ No. distinct threshold functions on N points in general position in K dimensions.

K=1 dimension, for any N

N points in a line: Right side origin - labelled 1

Left side origin - labelled 0

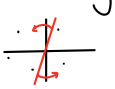
$\boxed{T(N,1) = 2}$ (distinct threshold functions)

N=1, for any K

Both possible labelling $\underline{w} = \pm \underline{x}^{(n)}$

$T(1,K) = 2$

K=2, Any N points



Free to Spin Seperating line around origin:

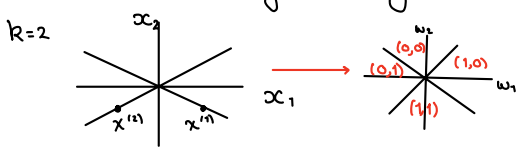
Each time passes point; new function

General position: No 2 points crossed at one time.

Each point passed over twice: $\boxed{T(N,2) = 2N}$

2D Input Space becomes 2D weight Space

$T(N,K)$ - No. distinct regions in weight Space



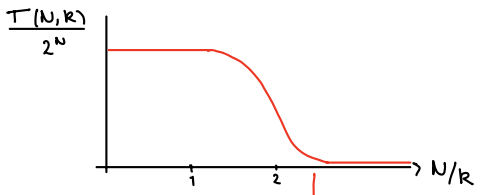
In general recursion relation:

$$\boxed{T(N,K) = T(N-1,K) + T(N-1,K-1)}$$

↳ Pascal: $C(N,K) = \binom{N}{K} = \frac{N!}{(N-K)!K!}$

$$\boxed{T(N,K) = 2\sum_{k=0}^{K-1}\binom{N-1}{K} = \begin{cases} 2^N & K \geq N \\ 2\sum_{k=0}^{K-1}\binom{N-1}{K} & K < N \end{cases}}$$

N \ K	1	2	3	4	5	6
1	2	2	2	2	2	2
2	2	4	4			
3	2	6	8			
4	2	8				
5	2	10				
6	2	12				



$\frac{T(N,k)}{2^N}$ → Probability arbitary labelling $\{t_n\}_{n=1}^N$ Can be memorized by neuron

↳ Can memorise $N=2k$ labels 'perfectly' but we will fail to memorise more.

Feed Forward Deep Neural Networks

Types of Neural Network:

↳ General purpose Neural Networks for Supervised learning

Image processing neural networks (ie Covolution Neural Networks)

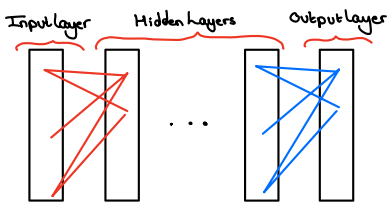
Sequential data neural networks (ie Recurrant Neural Networks)

Neural networks for unsupervised (ie Deep Boltzmann Machines)

Basics unit: 'Neutron'

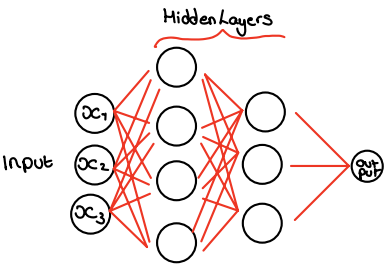
↳ Takes vector d input features: $x = (x_1, x_2, \dots, x_d)$

Produces Scalar output $a_i(x)$



Neural Network: Many neurons stacked into layers

↳ Output layer n → Input layer $n+1$



Function a_i

↳ Decomposed into

1) a linear operation that weights, w, relative importance of inputs.

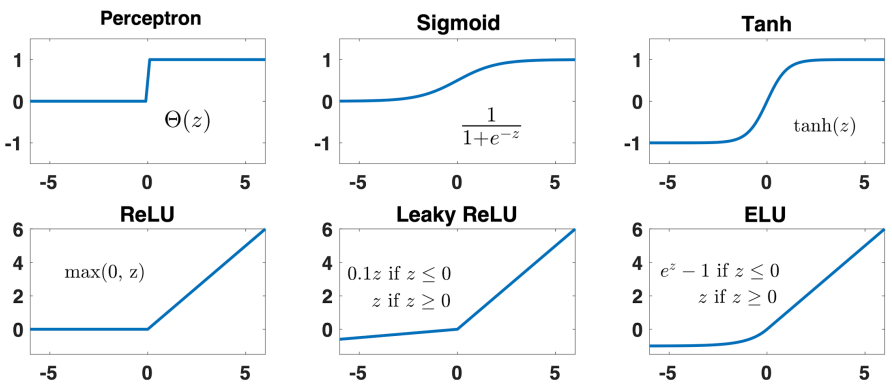
Weights: $w^{(i)} = (w_1^{(i)}, w_2^{(i)}, \dots, w_d^{(i)})$, Neuron Specific bias $b^{(i)}$

$$z^{(i)} = w^{(i)} \cdot x + b^{(i)}$$

2) Non-linear transformation $\sigma_i(z)$ - Same for all neurons.

$$a_i = \sigma_i(z^{(i)})$$

Examples Non-linearities



Neural Networks are trained using gradient descent:

↳ require derivatives of neural input-output function wrt weights $w^{(i)}$ and bias $b^{(i)}$

Perceptron: Discontinuous

Zero gradient everywhere $x \neq 0$

Cannot train using GD

Tanh/Sigmoid: Saturating activation function.

$\partial\sigma/\partial z \to 0$ for $z \gg 1$

Deep Networks (Layering Neurons)

↳ Neural network → layer neurons in hierarchical

Feed forward Neural Network

Neurons in Input layer: Inputs $\underline{x}$ → $a_i(\underline{x})$

Next layer: Outputs of previous layer becomes input of next layer

Output layer: Outputs Simple Classifier

- Discrete Labels: Logistic Regression / Softmaxe
- Continuous Labels: Linear Regression

Overall

Complicated non linear transform

↳ Depends on all weights and biases of all Neurons

↳ Input, Hidden, Output.

Advantages of Neurons

↳ Increases representational power (expressitivity)

↳ **Universal Approximation Theorem**

Neural network with a single hidden layer can approximate any continuous, multi-input/output function with arbitary accuracy.

↳ Each neuron generates activation functions with arbitary offsets and heights → added to form approx arbitary functions

#AskTomo

Modern Neural Networks

↳ Often contain many hidden layers → 'Deep'

More layers → More parameters → Higher representational power (expressitivity)

→ Learn more complex features from data.

↳ First layers: Simple features → Deeper Layers: more abstract features

Deep vs Shallow + Wide Networks

(Potentially) Computationally and algorithmically easier to train deep networks.

→ **Optimal Architecture**

↳ No. Layers and No. Neurons in each layer

Extensive experimentation and intuition

Problem Specific

→ Task, data amount, type, Computational resources

Training Deep Networks

1) **Loss function:**

For given data point $(\underline{x}_i, y_i)$, $\underline{x}_i \in \mathbb{R}^{d+1}$

Neural network prediction: $\hat{y}_i(\underline{w})$ → $\underline{w}$ parameters Neural network.

Output: Either Continuous predictor or Catagorical discrete predictor

Continuous Data (Same as linear regression)

↳ **Mean Squared Error:** $E(\underline{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i(\underline{w}))^2$

Mean absolute Error: $E(\underline{w}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i(\underline{w})|$

(L_1 norm)

Both: Often implement regularization

L_1, L_2 regularizers

Categorical Data

↳ Cross Entropy

Logistic classifier for binary data (True labels $y_i = \{0,1\}$)

Output: Probability x_i is predicted to be in category 1

$$\hat{y}_i(\underline{w}) = p(y_i = 1 | \underline{x}_i; \underline{w})$$

Cross Entropy

$$E(\underline{w}) = -\sum_{i=1}^{n} y_i \log\{\hat{y}_i(\underline{w})\} + (1-y_i)\log\{1-\hat{y}_i(\underline{w})\}$$

Softmax for greater than 2 labels (True labels $y_i \in \{0, 1, \ldots, M-1\}$)

Classified $y_{i,m} = \begin{cases} 1 & \text{if } y_i = m \\ 0 & \text{otherwise} \end{cases}$

Output: Probability that Neural network assigns data point to Category m

$$\hat{y}_{im}(\underline{w}) = p(y_i = m | x_i; w)$$

Cross Entropy

$$E(\underline{w}) = -\sum_{i=1}^{n} \sum_{m=0}^{M-1} y_{im} \log(\hat{y}_{im}(\underline{w})) + (1-y_{im}) \log[1-\hat{y}_{im}(\underline{w})]$$

↳ often Supplemented by additional terms that implement regularisation.

2) Optimisation of Cost Function

↳ Gradient descent based methods ($\underline{w}$ updated to move in direction $\nabla_w E(w)$)

Requires Back propogation

3) Back propogation (Additional Step for Neural Networks)

↳ Required to efficiently calculate derivative Cost function w.r.t every parameter in Neural network (weights + biases of all neurons in all layers)

L-Layers in Network: $l = 1, \ldots, L$

w_{jk}^l → The weight for the connection from k^{th} neuron in layer $l-1$ → j^{th} neuron in layer l

b_j^l → Bias for j^{th} neuron in l^{th} layer.

Activation a_j^l: j^{th} neuron in l^{th} layer → related to activities of neurons in layer $l-1$ (feed forward neural network).

$$a_j^l = \sigma\left(\sum w_{jk}^l a_k^{l-1} + b_j^l\right) = \sigma(z_j^l) \quad \rightarrow \text{Def: } z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

↳ Cost function: Dependent on a_j^L - activities of output layer

Error (j^{th} Neuron L^{th} (output) layer)

Change in cost function wrt weighted input z_j^L → Error of given neuron j in Layer L

$$\Delta_j^L = \frac{\partial E}{\partial z_j^L}$$

1) $$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} = \frac{\partial E}{\partial a_j^l} \cdot \sigma'(z_j^l)$$

Overall Error

depends on Neurons in layer L through activation of Neurons in Layer $l+1$: Δ_k^{l+1}

$$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \sum_k \frac{\partial E}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

$$= \sum_k \Delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \left(\sum_k \Delta_k^{l+1} w_{kj}^{l+1}\right) \sigma'(z_j^l)$$

Similiarly wrt bias

2) $$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial b_j^l}\underbrace{\frac{\partial b_j^l}{\partial z_j^l}}_{1} = \frac{\partial E}{\partial b_j^l}$$

3) $$\Delta_j^l = \left(\sum_k \Delta_k^{l+1} w_{kj}^{l+1}\right) \sigma'(z_j^l)$$

Overall: $\frac{\partial E}{\partial w^l_{jk}} = \frac{\partial E}{\partial z^l_j} \times \frac{\partial z^l_j}{\partial w^l_{jk}} = \Delta^l_j \, a^{l-1}_k$

Together: Back propogation Algorithm

1) Activation of Input layer
   - Calculate a^1_j of all neurons (in input layer (=1)

2) Feed forward:
   - Compute z^l and a^l for each subsequent layer
     - exploiting feed-forward architecture.

3) Error at toplayer
   - Calculate using ①

   Requires Cost function: $E(\underline{w}) = E(\underline{a}^L)$ and activation function $\sigma(z)$

4) 'Back propogate' error
   - Calculate Δ^L_j for all layers (working backwards)

5) Calculate gradient

   Calculate $\frac{\partial E}{\partial b^l_j}$, $\frac{\partial E}{\partial w^l_{jk}}$

Computationally efficient
- Essential as for each step of GD must know derivative wrt every parameter.

ISSues in training DNN
- Problem of vanishing or exploding gradients

often found in Neural Nets capturing long-range dependencies
- ie Recurrent Neural Networks for sequential data

Example: Single neuron per layer
All weights equal, w

$$\Delta^1_j = \Delta^L_j \prod_{j=0}^{L-1} w\sigma'(z_j) = \Delta^L_j (w)^L \prod_{j=0}^{L-1} \sigma'(z_j)$$

Assume $\sigma'(z_j) \approx$ Const $= \sigma'_0$

$$\Delta^1_j = \Delta^L_j (w\sigma'_0)^L$$

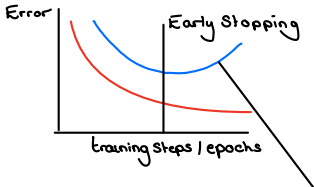
- $w\sigma'_0 > 1$: Errors and Gradients blow up
- $w\sigma'_0 < 1$: Errors and Gradients vanish

When weights satisfy: $w\sigma'_0 \approx 1$

Avoided by: Initialising weights of GD cleverly
'Gradient Clipping'

L_1 / L_2 Penalties
- Implemented by Keras

DNN's Bias variance Trade off



Early Stopping:

Divide training data into two → training, Smaller validation set
- out of sample performance

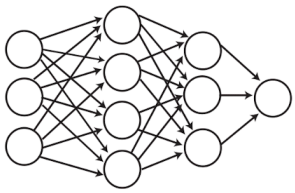
Use: ModelCheckpoint

Increase due to overfitting

Dropout

↳ reduce Spurious Correlations between Neurons.

Randomly drops out neurons (along with connections) temporarily during each Step of the training → given neuron can not be relied on.

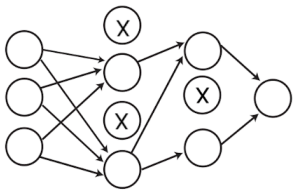


↳ Training

For each mini batch in Grad Desc Step

↳ Neuron dropped within Neural net with probability, p

Grad Desc performed on 'thinned' network.



Test

All neurons active.

Overall weights: average weights over all possible thinned neural networks

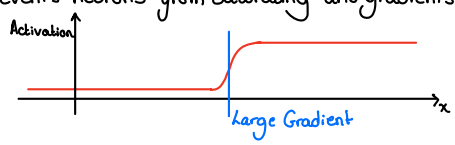
On average weights only present 'p' of the time.

↳ $W_{test} = p\, W_{train}$

Batch Normalisation

Observed: training Neural networks works best when inputs are Centered around zero with respect to bias.

↳ Prevents neurons from Saturating and gradients vanishing



Without: Changes in lower level parameters

↓

Saturation effects in higher layers (+Vanishing gradients)

Solution: Add new 'Batch Norm' layers that Standardize the inputs by the mean and varance of minibatch

For layer L with d neurons with inputs $(z_1^l, \dots, z_d^l)$

↳

$$z_k^l \longrightarrow \hat{z}_k^l = \frac{z_k^l - \mathbb{E}[z_k^l]}{\sqrt{Var(z_k^l)}}$$

→ $\mathbb{E}[z_k^l]$, $Var[z_k^l]$

↳ Calculated over minibatch:

ie $\mathbb{E}[z_i] = \sum_{i=1}^{N_{batch}} \frac{z_i}{N}$ N_{batch} - length of mini batch

Introduce Scaling Normalised Input → Effectively another layer that can be included in Back propogation

$$\hat{z}_k^l \longrightarrow \tilde{z}_k^l = \gamma_k^l \hat{z}_k^l + \beta_k^l$$

Significantly Increases Compute time (Speedsup Convergence)

Acts as powerful Regulariser

↳ Not fully understood

Maybe Single sample in different mini-batch → Introduces randomness.

Overall Process

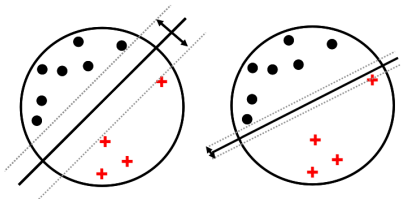
1) Collect and pre-process the data
2) Define model and architecture
3) Chose the cost function and optimizer
4) Train the model
5) Evaluate and study model performance on test data
6) Use validation data adjust hyperparameters and network architecture to optimize performance.
   - → 'Optuna'

Support Vector Machines

Large Margin Classifiers

Consider binary classification: $h(\underline{x}) = \text{Sign}(y(\underline{x}))$

With decision boundary $y(\underline{x}) = \underline{w}^T \underline{x} + w_0$



'Best margin': Maximum margin
- → Seperate classes

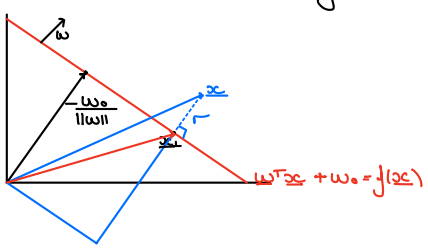
but also maximises distance of closest point to decision boundary

Comparison with perceptron

Perceptron ⟶ uses all datapoints to optimise parameters

Svm ⟶ only uses difficult points/outliers.

Distance to decision boundary



Origin to line: $\lambda \frac{w}{\|w\|}$ $\quad w^T \cdot \lambda \frac{w}{\|w\|} + w_0 = \lambda \frac{\|w\|^2}{\|w\|} + w_0 = 0$

$\lambda = -\frac{w_0}{\|w\|}$ → -ve by definition

$w^T \cdot \underline{x} = -w_0$

$\underline{x} = \underline{x}_\perp + r \frac{w}{\|w\|}$ $\quad x_\perp$ - x projected onto boundary, r - distance

$x_\perp$ lies on boundary: $w^T \cdot x_\perp + w_0 = 0$ → Plane defined by: $y(x) = 0$

$$w^T \left(\underline{x} - r \frac{w}{\|w\|} \right) + w_0 = 0$$

$$w^T \cdot \underline{x} - r\|w\| + w_0 = 0$$

$r = \frac{w^T \cdot \underline{x} + w_0}{\|w\|} = \frac{y(\underline{x})}{\|w\|}$ → The distance of point $\underline{x}$ to hyperplane $(\underline{w}\underline{x} + w_0 = 0)$ given by r:

Mathematical Inequality enforcing on Correct Side of boundary:

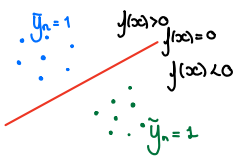
Label for classes: $\tilde{y}_n = \pm 1$

With boundary defined by: $y(x_n) = 0$

→ On one Side: $y(x_n) > 0$ → position so Side with $\tilde{y}_n = +1$

On Other Side: $y(x_n) < 0$ → position so Side with $\tilde{y}_n = -1$

If all on right Side $\quad y(x_n)\tilde{y}_n > 0$



Maximise distance to closest point: (Constrained optimization problem)

$$\max_{w, w_0} \frac{1}{\|w\|} \min_{n=1,\dots,N} [\tilde{y}_n (w^T x_n + w_0)]$$

- $\frac{1}{\|w\|}$: Norm vector w
- $\tilde{y}_n (w^T x_n + w_0)$: Signed distance to decision boundary

→ **Minimisation over n:** Finds closest point to decision tree using signed distance

Enforcing margin maximisation to take 'most limiting value'.

↳ Helps identifies 'support vectors'

Maximisation over w, w_0: Push boundary as far as possible from 'support vectors' in both classes

Note: Maximising $\frac{1}{\|w\|}$ → Same as maximising $\|w\|$

or even Minimizing $\frac{1}{2}\|w\|^2$

Simplifying with rescaling

By rescaling $w \to kw$, $w_0 \to kw_0$

No effect on distance, r and optimisation

∴ Define Scale factor S.t:

(for point closest to decision boundary) $\tilde{y}_n f(x_n) = 1$

Thus for all points:

$$\tilde{y}_n f(x_n) = \tilde{y}_n (w^T x_n + b) \geq 1 \quad \text{for all } n$$

Thus:

$$\max_{w, w_0} \frac{1}{\|w\|} \min_{n=1,\dots,N} [\tilde{y}_n (w^T x_n + w_0)] \to \min_{w, w_0} \frac{1}{2}\|w\|^2 \quad \text{S.t} \quad \tilde{y}_n (w^T x_n + w_0) \geq 1, \; n = 1, \dots, N$$

(Optimization; Constraint)

↳ Constrained optimisation.

Lagrangian Multipliers: Optimisation

$$\mathcal{L}(w, w_0, \alpha) = \frac{1}{2} w^T w - \sum_{n=1}^{N} \alpha_n (\tilde{y}_n (w^T x_n + w_0) - 1)$$

↓

Seek: $(w, w_0, \alpha) = \min_{w, w_0} \max_{\alpha} \mathcal{L}(w, w, \alpha)$

$$\nabla_w \mathcal{L}(w, w_0, \alpha) = w - \sum_{n=1}^{N} \alpha_n \tilde{y}_n x_n = 0 \to \underline{w = \sum_{n=1}^{N} \alpha_n \tilde{y}_n x_n}$$

$$\frac{\partial}{\partial w_0} \mathcal{L}(w, w_0, \alpha) = -\sum_{n=1}^{N} \alpha_n \tilde{y}_n = 0 \to \underline{\sum_{n=1}^{N} \alpha_n \tilde{y}_n = 0}$$

Note: Scaling

↳ Scaling is important to ensure maximal margin

↳ If dimensions have different scales - 1 will dominate in $\|w\|$

$$x' = \frac{x - \mu}{\sigma}$$

Substituting

$$\mathcal{L}(\hat{w}, \hat{w}_0, \alpha) = \frac{1}{2} w^T \hat{w} - \sum_{n=1}^{N} \alpha_n \tilde{y}_n w^T x_n - \sum_{n=1}^{N} \alpha_n \tilde{y}_n w_0 + \sum_{n=1}^{N} \alpha_n$$

$$= \frac{1}{2} \hat{w}^T \hat{w} - \hat{w}^T \hat{w} - 0 + \sum_{n=1}^{N} \alpha_n$$

$$= \frac{1}{2} \hat{w}^T \hat{w} - \hat{w}^T \hat{w} - 0 + \sum_{n=1}^{N} \alpha_n$$

$$= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j x_i^T x_j + \sum_{n=1}^{N} \alpha_n \to$$

Wish to maximise w.r.t α Subject to Constraints

$\sum_{n=1}^{N} \alpha_n \tilde{y}_n = 0$ and $0 \leq \alpha_n$ for $n = 1, \dots, N$

In practise better results by: averaging over all Support Vectors

Soft margin Classifiers

↳ Helps to achieve classification 'sensibly'

Can classify non-linearly seperable data (without Kernels) by allowing error.

↳ If non linearly seperable: no solution $\tilde{y}_n f(x_n) \geq 1$ for $n=1,\dots,N$

Introduce: Slack variables: $\xi_n \geq 0$

Soft margin constraints: $\tilde{y}_n f(x_n) \geq 1 - \xi_n$ (rather than $\tilde{y}_n f(x_n) \geq 0$)

New objective: $\min_{w, w_0} \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N}\xi \quad$ s.t $\quad \xi_n \geq 0, \quad \tilde{y}_n(\underline{x}_n^T \underline{w} + w_0) \geq 1 + \xi_n$

↳ C, hyperparameter - defining how many points we allow to violate the margin constraint ⟶ $C = \infty$ - unregularised, hard margin classifier

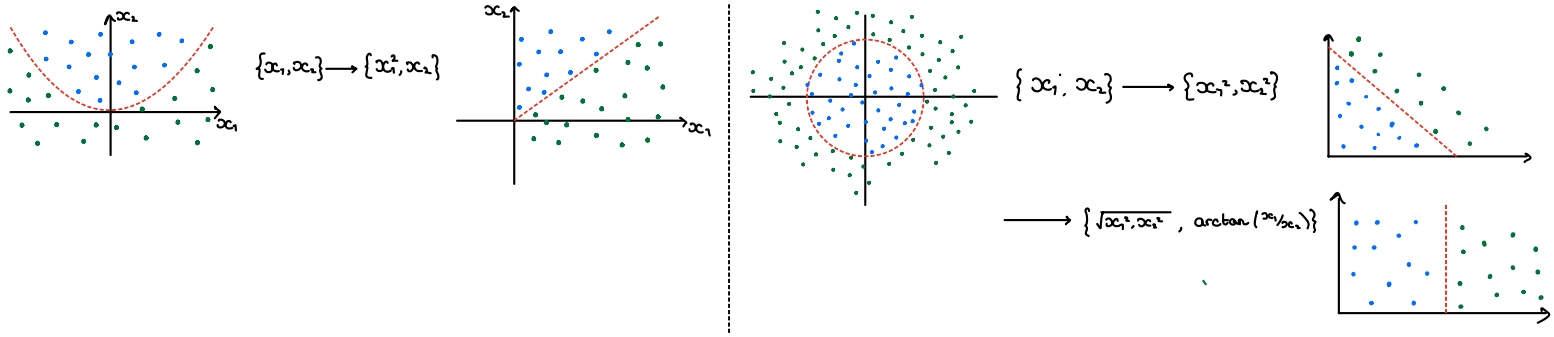
Corresponding Lagrangian: $L(\underline{w}, w_0, \underline{\alpha}, \underline{\xi}, \underline{\mu}) = \frac{1}{2}\underline{w}^T\underline{w} + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}\alpha_n(\tilde{y}_n(\underline{w}^T\underline{x}_n + w_0) - 1 + \xi_n) - \sum_{n=1}^{N}\mu_n\xi_n \qquad \alpha_n > 0, \ \mu_n > 0$

Kernel Trick

↳ Allows us to handle data that is not linearly seperable in its own feature space.

Try transforming into higher dimensional space where linearly seperable

ie:



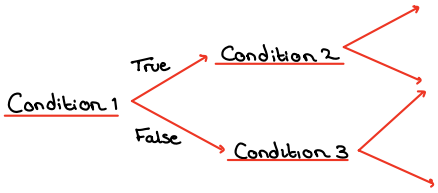
Option 1: Explicitly map into higher dimensional space, $\phi(x)$

Computationally expensive, infeasible

Option 2: Use Kernel function $K(x, x')$ - implicitly calculates dot product in higher dimensional space. $K(x, x') = \phi(x) \cdot \phi(x')$

$$f(x) = \hat{w}^T x + \hat{w}_0 = \sum \alpha_n \tilde{y}_n \underline{x}_n^T \underline{x} + \hat{w}_0 \longrightarrow \boxed{\sum \alpha_n \tilde{y}_n K(\underline{x}_n, \underline{x}) + w_0}$$

Decision Trees



For a given training dataset:
- Wish to find best tree structure
  ie order of features, decisions.

Described by: Depth of tree → favour Symmetric tree
- Maximum no. splitting conditions → maximum 'gain'

Wish to maximise 'information gain' per split.

$$\text{Gain}(S,A) = \text{Entropy}(S) - \sum_{v \in A} \underbrace{\frac{|S^v|}{|S|}}_{\text{weight of subset}} \text{Entropy}(S^v)$$

$$\text{Entropy}(S) = -\sum_{i=1}^{n} p_i \log p_i$$

- Wish to maximise Gain.

S - Set, S^v - Set with attribute $v \in A$

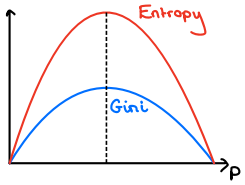
A - Attributes

p_i - Proportion of samples in S that belong to class i

Alternatively

$$\text{Gain}(S,A) = \text{Gini}(S) - \sum_{v \in A} \frac{|S^v|}{|S|} \text{Gini}(S^v)$$

$$\text{Gini}(S) = 1 - \sum_{i=1}^{n} p_i^2 \xrightarrow{\text{2 classes}} 2p^2 + 2p$$



Day	Outlook	Temp.	Humidity	Wind	Go hiking?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

$$\text{Gain}(S,\text{Outlook}) = \text{Entropy}(14) - \frac{5}{14}E(\text{outlook}=\text{Sunny}) - \frac{4}{14}E(\text{outlook}=\text{Overcast}) - \frac{5}{14}E(\text{outlook}=\text{rainy})$$

$$\text{Entropy}(14) = -\frac{9}{14}\log\left(\frac{9}{14}\right) - \frac{5}{14}\log\left(\frac{5}{14}\right)$$

$$\text{Entropy}(\text{outlook}=\text{Sunny}) = -\frac{2}{5}\log\left(\frac{2}{5}\right) - \frac{3}{5}\log\left(\frac{3}{5}\right)$$

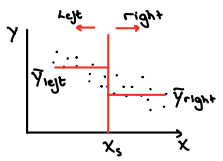
Iterate for all calculations:

Find: $\text{Gain}(S,\text{Outlook}) > \text{Gain}(S,\text{Temp}), \text{Gain}(S,\text{Humidity}), \text{Gain}(S,\text{Wind})$

$\therefore$ 'Outlook first' decision

Now treat results as new samples

Continuous variable



x_s - threshold of splitting data

$\bar{y}_{left}$ - average y value for data $x_i \leq x_s$

$\bar{y}_{right}$ - average y value for data $x_i > x_s$

Overfitting:

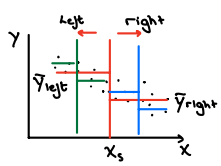
If tree too complex → risk overfitting to noise

Aim: find x_s that minimises error

$$\text{Error}_{left} = \frac{1}{|left|}\sum_{i \in left}(y_i - \bar{y}_{left})^2 \qquad \text{Error}_{right} = \frac{1}{|right|}\sum_{i \in right}(y_i - \bar{y}_{right})^2$$

$$L(x_s) = \frac{1}{|left|}\sum_{i \in left}(y_i - \bar{y}_{left})^2 + \frac{1}{|right|}\sum_{i \in right}(y_i - \bar{y}_{right})^2 \rightarrow \text{Minimise } L(x_s) \text{ w.r.t } x_s$$

Next Step



Ensemble Methods

Bias variance tradeoff (for ensembles)

↳ for limited data : often preferable to use less complex model with higher bias
less sensitive to sampling noise.

Ensemble Advantages

Statistical:

For Small learning Set:

Typically several models can be found - with similiar performance on training set.
Provided predictions are uncorrolated - averaging across models reduces risk of wrong / 'worst' hypothesis

Computational

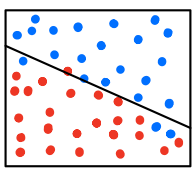
Learning algorithims subject to getting stuck in local optima.
Ensemble of models built with different starting points
Provide better approx of true unknown

Representational

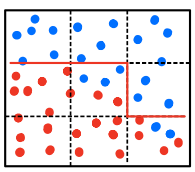
For learning set of fixed size
- often training data cannot be represented by single model.
Using multiple models — expand space of representable functions.

→ Reduce variance and easier to train individually

Linear perceptron



Ensemble linear models



Considerations: Randomise ensemble construction.
↳ Reduce corrolated errors

Bagging

For a very large dataset $\mathcal{L}$
↳ Partition into M smaller data sets $\{\mathcal{L}_1, \dots \mathcal{L}_M\}$
↳ Sufficiently large to learn predictor.

Train predictors on each data subset:
↳ Create 'ensemble aggregrate predictor' composed of them
ie For continuous predictor (ie regression)
↳ average of individual predictions

$$\hat{g}_{\mathcal{L}}^{A}(\underline{x}) = \frac{1}{M} \sum g_{\mathcal{L}_i}(\underline{x})$$

For classification tasks:
↳ Predictor: Class label $j \in \{1, \dots, J\}$
Majority decision by all predictors

$$g_{\mathcal{L}}^{A}(\underline{x}) = \underset{j}{\text{argmax}} \sum_{i}^{M} I[g(\mathcal{L}_i(\underline{x}) = j)]$$

→ Indicator function:

$$I[g_{\mathcal{L}_i}(x) = j] = 1 \text{ if } g_{\mathcal{L}_i}(x) = j$$

Requires: Sufficient data in each to set $\mathcal{L}_i$
↳ Otherwise: 'Empirical Bootstrapping' $\mathcal{L}_1 = \{x_i', \dots x_2, \dots x_1\}$ Random
↳ Sampling with replacement. $\{\mathcal{L}_1^{BS}, \dots, \mathcal{L}_M^{BS}\}$ - contain repeated points → $\hat{g}_{\mathcal{L}}^{BS}(\underline{x}) = \frac{1}{M} \sum_{i=1}^{M} g_{\mathcal{L}_i^{BS}}(\underline{x})$

Bootstrap Result:

Construct approx ensemble — reduce variance

Increase in bias of bagged estimator

Boosting

Changes how ensemble combined:

Ensemble 'weak' classifiers combined with aggregate boosted classifier.

↳ Each classifier - associated weight α_k - defines contribution

$$g_A(\underline{x}) = \sum_{k=1}^{M} \alpha_k g_k(\underline{x}) \quad \rightarrow \quad \Sigma \alpha_k = 1$$

Adaptive Boosting (adaBoosting)

↳ At each iteration: reweight error function to highlight data points where classifier preforms poorly.

For dataset: $\mathcal{L} = \{(\underline{x}_i, y_i), i = 1, .. N\}$ where $y_i \in \{-1, 1\}$

Classifier: $g : \underline{x} \rightarrow y$

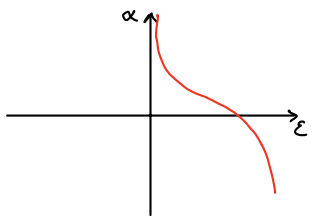
Set of classifiers $\mathcal{H} = \{g : x \rightarrow y\}$

Initialise: $w_{t=1}(\underline{x}_n) = \frac{1}{N}$, $n = 1, \ldots, N$

For $t = 1, \ldots, T$ (desired termination step)

↳ 1. Select Hypothesis that minimises weighted error

$$\epsilon_t = \sum_{t=1}^{N} w_t(\underline{x}_i) \, \mathbb{I}\left(g_t(\underline{x}_i) \neq y_i\right)$$



2. Let $\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$, update weight for each data $\underline{x}_n$ by:

$$w_{t+1}(x_n) \leftarrow w_t(x_n) \exp\left[-\alpha_t y_n g_t(\underline{x}_n)\right]$$

Where $Z_t = \sum_{n=1}^{N} w_t(\underline{x}_n) e^{-\alpha_t y_n g_t(\underline{x}_n)}$ (ensures weights sum to unity)

↳ highlights where aggregate classifier preforms poorly

Output: $g_A(\underline{x}) = \text{Sign}\left(\sum_{t=1}^{T} \alpha_t g_t(\underline{x})\right)$

Random Forests

Family of randomized tree-based classifier decision tree

↳ Decision trees — High variance, weak classifiers, easily randomized.

Creating ensemble decision trees:

↳ 1. 'bag' decision trees by training on different bootstrapped datasets.

2. 'feature bagging' - different random selection features at each split

↳ Reduces correlation between trees

Unsupervised Learning

↳ Discovering structures in unlabeled data

Uncover and exploit hidden structures in the data

Clustering

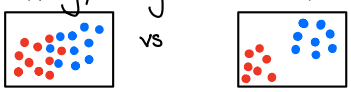
↳ Aim: group data into clusters by some similarity or distance measure

'Set of points sharing pattern or structure'

Common Considerations for Choosing method:

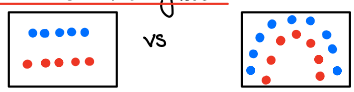
Distribution of Clusters

Overlapping / noisy vs well seperated



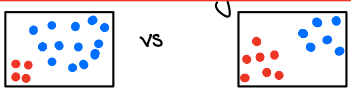
Geometry

flat vs non flat



Cluster Size

Multiple Sizes vs Uniform Sizes



Dimensionality

low vs high dimensionality

Computational Efficiency

Small vs large dataset

Note: Focus on 'Euclidean distance' as similarity measure

But similarity measures exist and can be more optimal for certain applications.

K-Means Clustering

↳ For N unlabelled observations $\{\underline{x}_n\}_{n=1}^{N}$ $\quad \underline{x}_n \in \mathbb{R}^p$ (p-no. features)

K 'Cluster means' (Centers): $\{\underline{\mu}_k\}_{k=1}^{K}$ with $\underline{\mu}_k \in \mathbb{R}^p$

↳ 'representatives of each cluster'

data points assigned to.

Given fixed no. K clusters:

find cluster means $\{\underline{\mu}\}$ and 'data point assignment' by minimising:

$$C(\{\underline{x}, \underline{\mu}\}) = \sum_{k=1}^{K} \sum_{n=1}^{N} r_{nk} (\underline{x}_n - \underline{\mu}_k)^2$$

Assignment: $r_{nk} = \begin{cases} 1 & \text{if } \underline{x}_n \text{ assigned to cluster } k \\ 0 & \end{cases}$ $\quad \sum_K r_{nk} = 1 \ \forall n$

$\sum_n r_{nk} = N_k$ → No. points assigned to cluster k.

i.e. find best cluster means that minimize variance of each cluster.

C - Sum of moments of inertia of every cluster

↳ Cluster mean ~ Centre of mass.

Kmeans Algorithm

Initialise: assign random cluster centers

Alternates between:

1. Expectation: Given set of assignments $\{r_{nk}\}$ minimise C with respect to $\underline{\mu}_k$ by solving derivative C wrt μ_k

$$\underline{\mu}_k = \frac{1}{N_k} \sum_n r_{nk} \underline{x}_n$$

2. Maximisation: Given Set of cluster means $\{\mu_k\}$, find assignments $\{r_{nk}\}$ which minimises C

Achieved by assigning each data point to nearest cluster mean.

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_{k'} (x_n - \mu_{k'})^2 \\ 0 & \text{Otherwise} \end{cases}$$

Termination: Change in objective function over iterations becomes smaller than pre-specified threshold.

Properties

↳ Guaranteed to Converge

C - local minimisation: Initialise across many random starting positions.

↳ Post select best local minimum.

Drawback: All cluster variances assumed the same.

If True clusters have very different variances (spreads) → errors

Agglomerative methods (bottom up approach)

↳ Stars with small initial clusters → progressively merged to form larger clusters

'Hierachy of clusters'

Use - 'distance measure between clusters $d(X,Y)$ for clusters X, Y

↳ two closest (according to distance measure) are merged until single is left

Agglomerative algorithm:

1. Initialize each point to its own cluster
2. Given a set of K clusters: $X_1, X_2, \dots, X_K$ merge until 1 cluster is left:
   a) Find the closest pair of clusters (x_i, x_j):
   $$(i,j) = \underset{i',j'}{\arg\min}\ d(x_{i'}, x_{j'})$$
   b) Merge the pair: Update: $K \leftarrow K-1$

Drawbacks: High computational complexity: $O(N^2)$

Not suitable large datasets

'Fix': Start with Kmeans and proceed hierarchica

Examples of 'distances' used:

The distance between clusters i and j defines by:

1. Single - linkage: Minimum distance between two elements of different clusters.
$$d(x_i, x_j) = \min_{x_i \in X_i, x_j \in X_j} \|x_i - x_j\|_2$$

2. Complete linkage: Maximum distance between two elements of different clusters.
$$d(x_i, x_j) = \max_{x_i \in X_i, x_j \in X_j} \|x_i - x_j\|_2$$

3. Average linkage: Average distance between points of different clusters
$$d(x_i, x_j) = \frac{1}{|x_i| \cdot |x_j|} \sum_{x_i \in x_i, x_j \in x_j} \|x_i - x_j\|_2$$

4. Wards linkage: Similiar to K-means - minimises 'total inerbia - 'error squared' before and after merging
$$d(x_i, x_j) = \frac{|x_i||x_j|}{|x_i \cup x_j|} (\mu_i - \mu_j)^2$$

Where μ_j is the centre of cluster j

Gaussian Mixture Models

Consider Clustering as learning most probable cluster identity (latent variable)

Must make assumption of how underlying distribution was generated. (Generative Model)

↳ eg. A Gaussian with some mean and variance.

GMM: points drawn from 1 of K Gaussians each with mean μ_k and Covariance matrix Σ_k

$$\mathcal{N}(\underline{x} \mid \mu, \Sigma) \sim \exp\left(-\frac{1}{2}(\underline{x}-\mu)\Sigma^{-1}(\underline{x}-\mu)^T\right)$$

π_k: Probability point drawn from mixture k

↳ Thought of as 'amplitude' of given distribution

$$p(z_k=1) = \pi_k$$

Probability of $\underline{x}$ being generated (by any cluster):

$$p(\underline{x} \mid \{\mu_k, \Sigma_k, \pi_k\}) = \sum_{k=1}^{K} \mathcal{N}(\underline{x} \mid \mu_k, \Sigma_k)\pi_k$$

Given dataset $X = \{\underline{x}_1, \dots, \underline{x}_N\}$;

Likelihood of data set:

$$p(X \mid \{\mu_k, \Sigma_k, \pi_k\}) = \prod_{i=1}^{N} p(\underline{x}_i \mid \{\mu_k, \Sigma_k, \pi_k\}) \qquad \theta: \text{parameters } \{\mu_k, \Sigma_k, \pi_k\}$$

Denote N latent variables for dataset X by Z $\qquad \underline{x} \rightleftarrows \underline{z}$ - binary latent Space.

For K-dimension binary Variables: $\underline{z} = (z_1, \dots, z_k, \dots z_K)$

$z_k = 1 \quad z_{i \neq k} = 0$ eg (1,0,0), (0,1,0) (0,0,1) K possible States

↳ Labels gaussian models to draw from (represent membership)

Probability $p(\underline{x} \mid \underline{z})$ of observing a data point $\underline{x}$ given $\underline{z}$ as:

↳ 'told what gaussian it includes'

$$p(\underline{x} \mid \underline{z}; \{\mu_k, \Sigma_k\}) = \prod_{k=1}^{K} \mathcal{N}(\underline{x} \mid \mu_k, \Sigma_k)^{z_k}$$ → fixed z - no need for π_k

Probability of observing given latent variable:

$$p(z_k = K) = \pi_k$$

Thus using Bayes' rule

Joint probability of Clustering assignment $\underline{z}$ and data point $\underline{x}$

given GMM parameters $\{\mu_k, \Sigma_k, \pi_k\}$

$$p(\underline{x}, \underline{z}; \theta) = p(\underline{x} \mid \underline{z}; \{\mu_k \Sigma_k\})\, p(\underline{z} \mid \{\pi_k\})$$ → more Simply $P(x) = \sum_{k=1}^{K} P(x \mid z=k)\, P(z=k)$

How to optimise parameters $\theta = \{\pi_k, \mu_k, \Sigma_k\}$

Given dataset $X = \{x_1, \ldots, x_N\}$ (independent samples)

Maximum likelihood estimate:

Log likelihood:

$$L = \log\left(P(\underline{X}\,|\,\{\pi_k, \mu_k, \Sigma_k\}\right) = \log\left(\prod_{i=1}^{N} P(\underline{x}_i\,|\,\{\pi_k, \mu_k, \Sigma_k\}\right) = \sum_{i=1}^{N} \log\left(P(\underline{x}_i\,|\,\{\pi_k, \mu_k, \Sigma_k\}\right)$$

↳ Iterate over datapoints

$$= \sum_{n=1}^{N} \log\left(\sum_{k=1}^{K} P(\underline{x}_n|\,\{\pi_k, \mu_k, \Sigma_k\}\right) = \sum_{n=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k\, \mathcal{N}(\underline{x}_n|\,\mu_k, \Sigma_k)\right)$$

↳ Iterate over Gaussian ↳ Convert to Gaussian dep. x weight

$$\boxed{L = \sum_{n=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k\, \mathcal{N}(\underline{x}_n|\,\mu_k, \Sigma_k)\right)}$$

Optimise this loss:

$$\boxed{\hat{\underline{\theta}} = \underset{\theta}{\operatorname{argmax}}\,(L)}$$

$$\frac{\partial L}{\partial \mu_k} = \sum_{n=1}^{N} \frac{\pi_k \frac{\partial}{\partial \mu_k}\mathcal{N}(\underline{x}_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j\, \mathcal{N}(\underline{x}_n|\mu_j, \Sigma_j)} = -\sum_{n=1}^{N} \frac{\pi_k\, \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum \pi_j\, \mathcal{N}(x_n|\mu_j, \Sigma_j)} \Sigma_k^{-1}(\underline{x}_n - \mu_k)$$

↳ Inverse Covariance Matrix

Responsibility Term, γ_{nk}

Responsibility, γ_{nk}

$$\gamma_{nk} = \frac{\pi_k\, \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum \pi_j\, \mathcal{N}(x_n|\mu_j, \Sigma_j)} = p\left(z_n^k\,|\,\underline{x}_n; \mu_k, \Sigma_k, \pi_k\right)$$

$$p(z_k=1|\,\underline{x}) = \frac{P(z_k=1)\,P(\underline{x}|\,z_k=1)}{\sum_{j=1}^{K} P(z_j=1)\,P(\underline{x}|\,z_j)} = \frac{\pi_k\, \mathcal{N}(\underline{x}|\,\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j\, \mathcal{N}(\underline{x}|\,\mu_j, \Sigma_j)}$$

→ 'Posterior probability once we observe x_n'

'responsibility of component k to explain observation, x_n'

$$\frac{\partial L}{\partial \mu_k} = -\sum_{n=1}^{N} \Sigma_K^{-1}(x_n - \mu_k)\,\gamma_{nk} = 0$$

Solving gives:

$$\mu_k = \frac{1}{\sum_n \gamma_{nk}} \sum_{n=1}^{N} \gamma_{nk} x_n \qquad \sum_n \gamma_{nk} = N_k$$

$$\boxed{\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk}\, x_n}$$

Similiarly: $\frac{\partial L}{\partial \Sigma_k} = 0 \rightarrow$

$$\boxed{\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk}\,(x_n - \mu_k)(x_n - \mu_k)^T}$$

For π_k: must do wrt constraint $\sum \pi_k = 1$

$$\tilde{L} = L + \lambda\left(\sum_{i=1} \pi_i - 1\right)$$

$$\frac{\partial \tilde{L}}{\partial k} = 0 \rightarrow \boxed{\pi_k = \frac{N_k}{N}}$$

γ_{nk} Still dependent on θ: Solve using iterative scheme

Calculate γ_{nk} values with $\mu_k^{(t)}$, $\Sigma_k^{(t)}$, $\pi_k^{(t)}$

Maximise by: $\boxed{\mu_k^{(t+1)} = \frac{\sum_i \gamma_{ik}^{(t)} x_i}{\sum_i \gamma_{ik}^{(t)}}}$ $\boxed{\Sigma_k^{(t+1)} = \frac{\sum_i^N \gamma_{ik}^{(t)} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_i \gamma_{ik}^{(t)}}}$ $\boxed{\pi_k^{(t+1)} = \frac{1}{N} \sum \gamma_{ik}^{(t)}}$

Principle Component Analysis, PCA

↳ Dimensional reduction

Performs an orthogonal transformations of data to find high variance directions.

↳ Directions with small variance: 'noisy' and can be removed.

Singular Value Decomposition

Aim: For m data points

From: $\{x^{(1)}, \dots, x^{(m)}\}$ in $\mathbb{R}^n$

Compress to: $\{c^{(1)}, \dots, c^{(m)}\}$ in $\mathbb{R}^l$ where $l < n$

Encoding $f(x) = c$ Decoding $x = g(c)$

To measure goodness of compression:

↳ how accurate reconstruction is: $\boxed{\|x - \tilde{x}\| \ll 1}$

Reconstruction: $\boxed{\tilde{x} = g(f(x))}$

(Encoding + Decoding)

Method: Matrix Multiplication

Reminder Diagonalisation

$A = U D V^T$ (A: $m \times n$; U: $m \times m$; V^T: $n \times n$)

U, left Singular matrix: ↳ eigenvectors of $(AA^T)_{m \times m}$

V, right Singular matrix ↳ eigenvectors of $(A^TA)_{n \times n}$

D Diagonal Matrix ↳ $\begin{pmatrix} \lambda_1 & 0 \\ 0 & \ddots \end{pmatrix}$

Specifically:

$g(c) = Dc$ (D: diag $n \times l$ matrix) $n \to l$ dim reduction

Conditions on D:

Columns of D to be orthogonal to each other and unit norm:

$\boxed{\Sigma d_{ik} d_{ij} = \delta_{kj}}$ → Diagonal Orthonormal

Cost:

Minimise difference between reconstructed vs original

$\boxed{\text{Argmin}_c \|x - g(c)\|^2}$ ($g(c) = \tilde{x}$)

$= \text{Argmin}_c (x - g(c))^T (x - g(c)) = \text{Argmin}_c \; x^Tx - 2x^Tg(c) - g^T(c)g(c)$

$g(c) = Dc$

$= \text{Argmin}_c -2x^Tg(c) + g^T(c)g(c) = \text{argmin}_c -2x^TDc + c^T \underbrace{D^TD}_{1} c$

$= \boxed{\text{Argmin}_c \; -2x^TDc + c^Tc}$

Solving: $\nabla(-2x^TDc + c^Tc) = 0$

$= 2D^Tx + 2c = 0 \quad c = D^Tx \rightarrow$ Encoder function: $f(x) = D^Tx$

Reconstruction $r(x) = g(f(x)) = DD^Tx$

Minimise:

$\|x - r(x)\|^2$ to Determine D

$$D^* = \underset{D}{\operatorname{argmin}} \sum_{i,j} (x_j^{(i)} - r(x^{(i)})_j)^2 \quad \text{Subject to } D^TD = 1$$

For $L=1$ (compress to single dimension): $D = (d_1, \ldots, d_n)^T = d$

$c = d^Tx \quad x \in \mathbb{R}^n$

$$d^* = \underset{d}{\operatorname{argmin}} \sum_i \|x^{(i)} - \underbrace{dd^Tx_i}_{\text{reconstructed}}\|^2 \qquad dd^T = 1 \ \|d\|^2 = 1$$

$$= \underset{d}{\operatorname{argmin}} \sum_i \|x^{(i)} - (d^Tx_i)d\|^2$$

$$= \underset{d}{\operatorname{argmin}} \sum_i \|x^{(i)} - (x^{(i)T}d)d\|^2$$

$$= \underset{d}{\operatorname{argmin}} \|X - Xdd^T\|^2$$

Using $\|A\|_F = \sqrt{\operatorname{Tr} AA^T}$ — Where $X \in \mathbb{R}^{m \times n}$ (n-dim, m data points)

$$= \underset{d}{\operatorname{argmin}} \left(\operatorname{Tr}\left((X - Xdd^T)^T(X - Xdd^T)\right)\right)$$

$$= \underset{d}{\operatorname{argmin}} -2\operatorname{Tr}(X^TXdd^T) + \underbrace{\operatorname{Tr}(dd^TX^TXdd^T)}_{\operatorname{Tr}(X^TXdd^{\cancel{T}}\overset{1}{\cancel{d}}d^T)}$$

Invariance of rotation of Trace

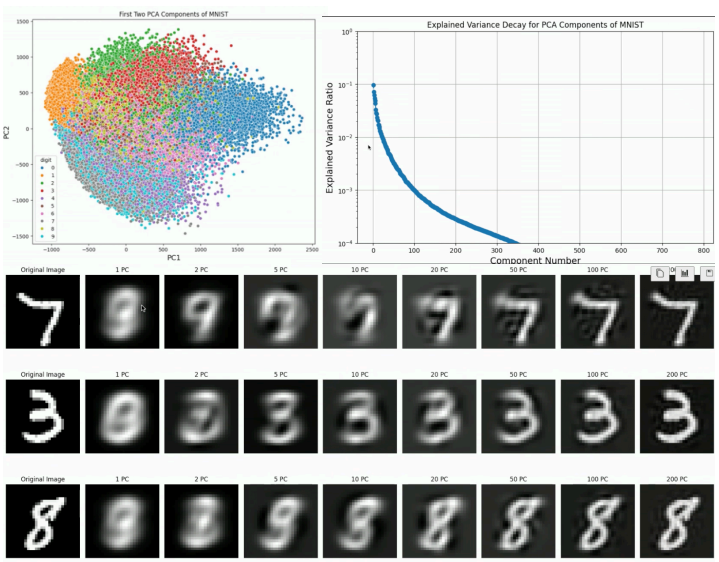
$$= \underset{d}{\operatorname{argmin}} -\operatorname{Tr}(X^TXdd^T) = \underset{d}{\operatorname{argmax}} \operatorname{Tr}(X^TXdd^T)$$

$$= \underset{d}{\operatorname{argmax}} \operatorname{Tr}(d^TX^TXd)$$

Optimal d given by eigenvector of XX^T corresponding to largest eigenvalue.

↳ For larger dimensions - iterates to next largest λ etc

PCA in Practise



Variance against dimensions:

As dimension increases - more agreement between x and $\bar{x}$

Variance decreases.

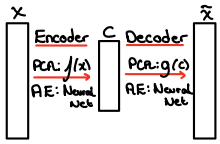
With few greater variance 'bub drop off is fast

Auto encoder

Similiar to PCA, but non linear

encoder and decoder

↳ More powerful insights



Issues Faced:

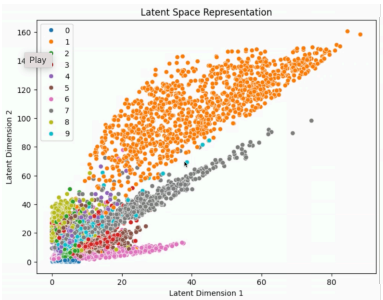
↳ Overfitting

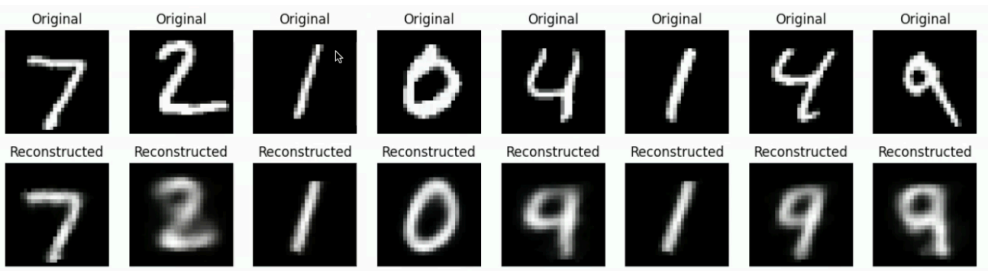
↳ regularisation, dropout.

Encourage model to have properties on latent dimensions (hidden features) → is the point not to discover these how can we encourage if we dont know what they are

↳ encourage meaniful representations.

Autoencoder in Practise:





Unsupervised Learning Approach

t-Stochastic Neighbour embedding (t-SNE)

Basic Idea of SNE

Conversion: High dim Euclidean Distances → Conditional probabilities 'of Similarities'

$$p(j|i) = \frac{\exp\left(-\frac{1}{2\sigma_i^2}\|x_i - x_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\frac{1}{2\sigma_i^2}\|x_i - x_k\|^2\right)}$$

Sum over data points.

Normalising factor

↳ likelihood x_j is x_i's neighbour

σ_i - variance datapoint i

↳ used for 'zooming' in denser regions

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2}$$

t-SNE

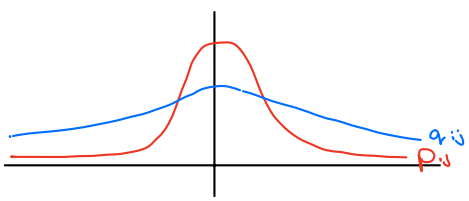
Similiar probability distribution in 'low dimensional latent space'

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq i} \left(1 + \|y_i - y_k\|^2\right)^{-1}}$$

$y_i \in \mathbb{R}^{p'}$ $p' < p$

↳ Long tail distribution, to preserve short distance information

But repels points far apart in original space.



Determine latent space coordinates y_i

Minimises K-L divergence between q_{ij} and p_{ij}:

$$C(Y) = D_{KL}(P\|Q) = \sum_{ij} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

Minimise with gradient descent:

$$\partial_{y_i} C = \sum_{j \neq i} 4 p_{ij} q_{ij} z_i (y_i - y_j) - \sum_{j \neq i} 4 q_{ij}^2 z_i (y_i - y_j) \quad \text{where } z_i = \frac{1}{\sum_{k \neq i} (1 + \|y_k - y_i\|^2)}$$

$$= F_{attractive} - F_{repulsive}$$

↳ Solution at equilibrium configuration.

of particle/embeddings y_i via these forces

Fixing σ_i with perplexity

σ_i - variance associated to each data point.

'Entropy': $H_i = -\sum_j p_{i|j} \log p_{i|j}$

'Perplexity': $2^{H_i} = P_i$ → Fix to constant.

↳ Hyper parameters.

Overall Properties

↳ Invariant under rotations in latent space

Stochastic Results: dependant on initial seed.

Preserves short-distance information.

Deforms scales