Algorithmic Trading

Arthur Li

December 13, 2025

Contents

1 Preface and Prerequisites

1.1 Brief Overview

Lorem Ipsum
To be completed once most of the book is done

1.2 Reading Roadmap

Lorem Ipsum. To be completed once most of the book is done

This content builds upon the foundational works of Rishi K. Narang (2013), Raja Velu (2020), and Marcos Lopez Prado (2018), among others, whose insights form the backbone of our discussion.

1.3 Overview of Systematic Investments

A schematic of a live 'production' trading strategy is shown below, but does not include everything else necessary to create the strategy (i.e., research tools).

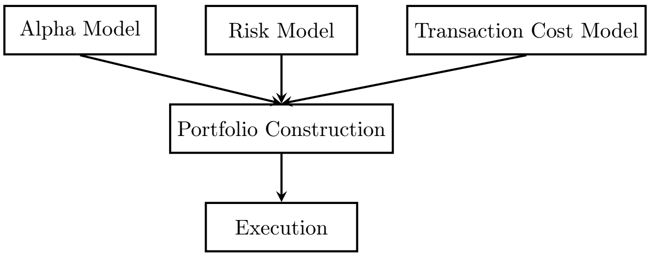

Figure 1: Live Production Trading Strategy Overview

At its core, the trading system is organised into three primary modules:
The trading system has three modules:

i. Alpha model: predicts the future of the instruments considered for trading, i.e. directional alpha

ii. Risk model: limits amount of exposure to factors that are unlikely to generate returns but could drive losses, i.e. directional exposure limit on an asset class

iii. Transaction cost model: determine if the cost of the trades needed to migrate from current portfolio to new portfolio is desirable to the portfolio construction model.

These models feed into a portfolio construction model that balances the tradeoffs of profit and risk to determine the best portfolio to hold. The model finds the differences in trades that need to be executed.
The execution model then takes the required trades, and using inputs such as urgency in which the trades need to be executed and dynamics of liquidity in the markets, executes the trades in an efficient and low cost manner.

Method 1.3.1. *Chains of Production for Alpha Signals*

i. Data Curation: for collecting, cleaning, indexing, storing, adjusting, and delivering all data to production chain. Requires experts in market microstructure and data protocols such as FIX.

ii. Feature Analysis: transform raw data into informative signals. Requires experts in information theory, signal extraction and processing, visualisation, labelling, weighting, classifiers, feature importance techniques. Feature analysts collect and catalogue libraries of findings.

iii. Strategists: informative features are transformed into actual investment algorithms. Strategists will parse libraries of features for ideas to develop an investment strategy. Require data scientists with deep knowledge of financial markets and economy. Features may be discovered by black box, but strategy is developed in a white box.

iv. Back-testers: assess profitability of investment strategy under various scenarios. Requires data scientists with deep understanding of empirical and experimental techniques. Good back-tested incorporates in analysis meta-information on how strategy was created.

v. Deployment Team: integrate strategy code into production line. Requires algorithm specialists and mathematical programmers. To ensure deployed solution is logically identical to prototype, and to optimise implementation sufficiently such that production latency is minimised.

vi. Portfolio Oversight: once strategy is deployed, follows lifecycle.

   1. Embargo: initially, strategy is run on data observed after end date of backtest. If embargoed performance is consistent with backtest, strategy is promoted to next stage.
   2. Paper Trading: strategy run on live, real-time feed. Performance accounts for data parsing latencies, calculation latencies, execution delays, and other time lapses between observation and positioning.
   3. Graduation: strategy manages real position, whether in isolation of as part of ensemble. Performance evaluated precisely, including attributed risk, returns, and costs.
   4. Re-allocation: based on production performance, allocation is re-assessed frequently and automatically. Strategy allocation follows a concave function, Initial allocation is small. As time passes and strategy performs as expected, allocation is increased. Over time, performance decays and allocations become gradually smaller.
   5. Decommission: if strategy perform below expectations for sufficiently extended period of time, strategy is discontinued.

1.3.1 Alpha Models Overview

Theory-driven models tests theories of why markets behave in a manner, and see if they can be used to predict the future. Strategies utilising price-related data are trend and mean reversion; strategies utilising fundamental data are value/yield, growth and quality. Usually more than one model is used in combination.

Definition 1.3.2. *Theory Driven Models*

i. Trend Following: markets move in given direction long enough that the trend can be identified. As more data support the bull/bear thesis in an uncertain market, more market participants will adopt the same thesis and hence move the asset price to a new equilibrium.
Moving average crossover indicator strategy has less than one point of return for every point of downside risk taken, as market behaviour are unstable and episodic.

ii. Mean Reversion: markets move in opposite direction to the prevailing trend. Short-term imbalances between buyers and sellers due to liquidity forces prices to move abruptly in one direction, which increases probability of trend reversion as liquidity issue is resolved.
Statistical arbitrage bets on convergence of prices of similar stocks whose prices have diverged.
Longer-term trends can occur despite smaller oscillations around these trends occurring in the shorter term, hence both strategies may be used in conjunction.

iii. Value/Yield: value strategies uses ratios of fundamental factor against the price of the instrument, inverted to keep the ratio consistent. The higher the yield, the cheaper the instrument.
Buying undervalued security and selling overvalued security is a *carry trade*. The difference between yield received and yield paid is the *carry*.
Quant Long Short (QLS) ranks stocks by attractiveness based on various factors such as value, then buy the higher-ranked stocks while shorting the lower-ranked stocks.

iv. Growth: make predictions based on asset's expected or historically observed level of economic growth. Forward-looking growth expectations are typically used as a metric.
Growth is trending, and strongest growers are becoming more dominant relative to competitors. Macro growth factors may be used on foreign exchange, while micro growth factors may be used on companies.

v. Quality: All else being equal, it is better to long high quality and short low quality. Capital safety is important. Factors include earnings quality, equity-to-debt ratios etc.

Data-driven models are more difficult to understand, with more complicated mathematics. Relies on data mining, more technically challenging and far less widely practiced. Typically more used in high-frequency space, as they can discern how market behaves without caring about the economic theory or rational.

Method 1.3.3. *Strategy Parameters*
An implementation approach requires a forecast target, time horizon, bet structure, investment universe, model specification, and run frequency.

i. Forecast Target: models may forecast direction, magnitude, duration of move, and may include probability into the forecast. Signal strength is of importance, defined by a larger expected return and/or higher likelihood of return. A higher level of signal strength results in a bigger bet taken on the position.

ii. Time Horizon: models may have forecast horizons ranging from microseconds to years. There are more variability between short-term and long-term strategies, as short-term strategies are making very large number of trades compared to long-term strategies.

iii. Bet Structure: models can be made to forecast an instrument relative in itself or to others. For relative forecasts, smaller clusters (pairs) or larger clusters (sectors) may be used. For pairs, few assets can be compared precisely and directly. Large cluster grouping may eliminate impact of general movement of the sector and hence focus on the relative movement of stocks within the sector, allowing for clearer distinction between group behaviour and idiosyncratic behaviour. Clusters may be created either via statistical methods or using heuristics (i.e., fundamentally defined industry groups).
Statistical methods may be fooled by data, leading to bad grouping. Heuristic grouping may be imprecise for conglomerates, and may be too rigid. Relative alpha strategies tend to exhibit smoother returns during normal times than intrinsic alpha strategies, but may face incorrect groupings during stressful periods. This may be mitigated by utilising several grouping techniques in concert.

iv. Investment Universe: choices made on geography, asset class, instrument class, and exclusions. Liquidity is preferred so estimations of transaction costs are reliable. Large quantities of high quality data is required, which is found in highly liquid and developed markets. Instruments with consistent behaviour is preferred, hence biotech stocks are excluded due to sudden, violent price changes. Hence, the most common asset classes and instruments modelled are common stocks, futures (on bonds and equity indices) and forex.

v. Model Specification: focuses on definition of the strategy mathematically, and may be the source of alpha. Specification details in terms of machine learning or data mining techniques are also defined, to assist in fitting models to the data and setting parameter values. Refitting frequency is also defined to refresh the model and make it adapt to current market conditions; may lead to greater risk of overfitting.

vi. Run Frequency: defined from monthly to real time frequency. Increasing frequency of runs lead to greater number of transactions and hence higher transaction costs, and risk of moving portfolio based on noisy data. Less frequency of runs lead to smaller number of larger-sized trades, hence may move the market with block trades; may also miss opportunities to trade at more favourable prices.

Method 1.3.4. *Blending of Models*
Most common approaches are linear models, nonlinear models, and machine learning models. If models are not combined, then several portfolios are constructed based on output from each model, then combined using portfolio construction techniques. The best method depends on the model.

i. Linear Models: require independence of factors, and each factor to be additive. To determine the weight of each alpha factor, multiple regression techniques may be used.

ii. Nonlinear Models: used when factors are not independent, or the relationship changes over time. Conditional models base the weight of one factor on the reading of another factor. Rotational models assign weights of factors that fluctuate over time based on updated calculations of the various signal's weights, giving higher weights to factors with better performance recently.

iii. Machine Learning Models: developing machine learning strategies takes as much effort to produce one true investment strategy as to produce a hundred. The complexities include data curation and processing, HPC infrastructure, software development, feature analysis, execution simulations, backtesting etc.
Decades ago, macroscopic alpha based on simple tools like econometrics are common, but this is quickly diminishing. Microscopic alpha however, becomes more abundant, but requires heavy ML tools.

1.3.2 Risk Models

Risk model concerns the intentional selection and sizing of exposures to improve the quality and consistency of returns. By pursing an alpha, we want to be invested in the movement of the exposure to profit in the long run.

Method 1.3.5. *Factor-Based Models*
Factor-based models decompose asset returns into contributions from systematic factors and idiosyncratic components. The most common factors include:

i. Market Factor: Captures the overall movement of the market.

ii. Size Factor: Reflects the differential risk associated with companies of varying market capitalizations.

iii. Value Factor: Accounts for risk due to discrepancies between market prices and fundamental valuations.

iv. Momentum Factor: Measures the tendency of asset prices to continue in their current trajectory.

This allows traders to understand which elements drive portfolio risk and adjust exposures accordingly.

Method 1.3.6. *Statistical Models*
Statistical risk models leverage historical data and probabilistic techniques to quantify risk parameters.

i. Historical Simulation: Directly computing risk metrics from past return distributions.

ii. Monte Carlo Simulation: Generating a large number of potential future return scenarios to estimate risk under diverse conditions.

iii. Parametric Methods: Employing analytical formulas based on assumed return distributions to calculate key risk measures.

These are useful for dynamically updating risk assessments as new market data become available.

Method 1.3.7. *Limiting Size of Risk*
The quantitative risk models that limit the size of risk varies by the manner in which size is limited, how risk is measured, and what is having its size limited.
Size limits can be limited by hard constraints and penalties. A hard limit may be arbitrary, hence penalty functions may be built to allow a position to increase beyond the limit level, only if the alpha model expects a significantly larger return. The levels of limits and penalties may be determined from either theory or data.
To measure risk, there are two methodologies. The first is longitudinal, and measures risk through the volatility of an instrument. The second is to measure the correlation or covariance between assets (dispersion).
Size limiting may be applied to single positions and groups of positions (sectors, asset classes). It may also be applied to various types of risks and the amount of portfolio leverage.

Method 1.3.8. *Limiting the Types of Risk*
To eliminate unintentional exposure as there is no expectation of being compensated sufficiently for accepting them. This can be achieved through theoretical or empirical risk models.

i. Theory-Driven Risk Models: focuses on systematic risk factors, derived from economic theory. Systematic risks cannot be diversified away. Equity may have market risk, sector risk, market capitalisation risk etc. Fixed income may have interest rate risk.

ii. Empirical Risk Models: uses historical data to determine the unnamed systematic risks that should be measured and mitigated. Uses principal component analysis (PCA) to discern unnamed systematic risks that may correspond to named risk factors. Used by statistical arbitrage traders who are betting on exactly the component of an asset's return not explained by systematic risks.

1.3.3 Transaction Cost Models

Trade is made only if it increases the odds or magnitude of return (from alpha model), or if it decreases the odds or magnitudes of loss (from risk model). However, this improvement should be higher than cost of trading. The transaction cost model is not designed to minimise cost of trading, only to inform portfolio construction engine the cost of making any given trade.

Remark 1.3.9. *Transaction Cost Components*

i. Commissions and Fees: paid to brokerages (access to other market participants), exchanges (improved transaction security) and regulators (operational infrastructure) for the services provided. The bank's infrastructure is used by quants, where the brokerage commissions are rather small on a per-trade basis. Brokers also collect clearing and settlement fees. Clearing is the activity involving regulatory reporting and monitoring, tax handling, and handling failure, taken place in advance of settlement. Settlement is the delivery of securities in exchange for payment in full.

ii. Slippage: the change in price between the time the quant system decides to transact and the time when the order is at the exchange for execution. Trend-following strategies suffer most from slippage as assets are already moving in desired direction; mean-reverting strategies suffer the least from slippage. The lower the latency to market, the smaller the slippage. The more volatile an asset, the bigger the slippage.

iii. Market Impact: measures how much an order moves the market by its demand for liquidity. The impact of the trade on the market is unknown until the trade has already been completed. There may also be interaction between slippage and market impact (i.e., selling when a stock is trending upwards).

Definition 1.3.10. *Types of Transaction Cost Models*

i. Flat Model: cost of trading is the same, regardless of size of order. Model is reasonable if size traded is nearly always about the same, and liquidity remains sufficiently constant.

ii. Linear Model: cost of trading increases at a constant rate relative to size of order. Better estimate than flat transaction cost model.

iii. Piece-Wise Linear Model: using piece-wise linear functions to model costs. Balance between simplicity and accuracy; better accuracy than flat or linear models.

iv. Quadratic Model: most computationally intensive, but also most accurate.

1.3.4 Portfolio Construction Models

Comes in two major forms: rule-based, optimisers. Rule-based models are based on heuristics, can be exceedingly simple or rather complex, and derived from human experience (trial and error). Optimisers comprises of an objective function and uses algorithms to reach the end goal.

Definition 1.3.11. *Rule-Based Models*

i. Equal Position Weighting: used if portfolio manager believes that if a position is good enough to own, no other information is needed in determining its size. Strength of signal is not used as input in weighting.
Model assumes that there is sufficient statistical strength and power to predict not only direction but also magnitude relative to other forecasts in the portfolio. Portfolio takes few large bets on 'best' forecast, many smaller bets on less dramatic forecasts; may take excess risk in an idiosyncratic event on a seemingly attractive position, resulting in adverse selection bias.

ii. Equal Risk Weighting: adjust position sizes inversely to volatilities or a measure of risk. More volatile positions given smaller allocations, less volatile positions given larger allocations.
When unit of risk is equalised, it is almost always a backward-looking measurement such as volatility. If volatility changes with time, then model will be misled.

iii. Alpha-Driven Weighting: position size based primarily on alpha model. Alpha signal determines size of position, but usually with size limits. Constraints used also includes limits on size of total bet on a group. May also have a function that relates the magnitude of forecast to size of position.
If model used in futures trend following, might suffer sharp drawdowns. Reliance on accuracy of alpha.

iv. Decision-Tree Weighting: decision path to arrive at the allocation for given instrument, depending on type of alpha model and type of instrument. Constraints may include percentage limits for allocation.
Model size grows dramatically if more alpha models or mode types of positions are included.

Remark 1.3.12. *Optimisers Models Parameters*
Harry Markowitz's mean variance optimisation (MVO) as the pioneer model. Models are based on principles of modern portfolio theory (MPT). Inputs include asset expected return (mean), asset variance, expected correlation matrix. Other inputs include size of portfolio in currency terms, desired risk level (volatility or expected drawdown), and other constraints such as liquidity, universe limits.
Model uses an objective function and an algorithm to seek the goal, usually maximising return of portfolio relative to volatility of portfolio returns.

i. Expected Return: alpha models as basis of expected return, which also includes expected direction.

ii. Expected Volatility: stochastic volatility forecasting methods is commonly used, as volatility may have high and low periods, with occasional jumps. GARCH model is most used.

iii. Expected Correlation: as instrument correlations are not stable over time, it is more appropriate to group assets together before computing correlation within the group.

Method 1.3.13. *Optimisation Techniques*

i. Unconstrained Optimisation: most basic form with no constraints. Might provide a single-instrument portfolio, where all money will be invested in instrument with highest risk-adjusted return.

ii. Constrained optimisation: constraints include position limits, limits on various groupings of instruments. Might result in constraints driving the portfolio construction more than the optimiser.

iii. Black-Litterman Optimisation: blends investor expectations with a degree of confidence about those expectations, and these with historical precedent evident in the data. Adjusts historically observed correlation levels by utilising investor's forecast of return for the various instruments.

iv. Grinold and Kahn's Approach: builds a portfolio of signals, instead of sizing positions. To build factor portfolios, each of which are usually rule-based portfolios based on a single type of alpha forecast. Each portfolio backtested, then series of returns are then treated as instruments of a portfolio by the optimiser. Number of factor portfolios is more manageable, usually not more than 20. What is optimised is then a handful of factor portfolios. The model allows for inclusion of risk model, transaction cost model, portfolio size, and risk targets as inputs.

v. Resampled Efficiency: to improve the inputs to optimisation by addressing oversensitivity to estimation error. To resample data using Monte Carlo simulation to reduce estimation error in inputs to the optimiser.

vi. Data-Mining Approaches: machine learning techniques such as supervised learning or genetic algorithms used, as MVO involves searching many possible portfolios to find the best.

1.3.5 Execution Model

Two basic ways to execute trade: through electronic, or through human intermediary. For electronic execution, achieved through direct market access (DMA), which allows traders to utilise the infrastructure and exchange connectivity of brokerage firms to trade directly on electronic markets.
Execution algorithms can be acquired through building, using broker's, or a third-party software vendors.
Brokerages offer portfolio bidding, where the 'blind' portfolio for transaction is described by characteristics such as valuation ratios of longs and shorts, sector breakdown, market capitalisation etc. Broker then quote a fee in basis points in terms of the gross market value of portfolio traded. Hence, certainty is provided by the broker to the trader. Once agreement reached, broker receives fee and assumes risk of trading out the portfolio at future market prices, which may be better or worse than prices guaranteed.

Remark 1.3.14. *Order Execution Algorithm Parameters*

i. Aggressive vs Passive: algorithm make decision of passive vs aggressive order, depending on how immediately the trader wants to do the trade. Market orders are considered aggressive. Limit order at current best order is fairly aggressive, while limit order below current bid is passive.
Many exchanges pay providers of liquidity for placing passive orders, charging traders for using liquidity provided. Orders that cross the spread are using liquidity by using a passive order placed by another

trader, reducing liquidity available. Paying for liquidity sweetens deal for passive order, only if order is actually executed; passive trader gets better transaction price and a commission rebate from the exchange. Momentum strategies uses more aggressive orders; mean reversion uses more passive orders. A stronger, more certain signal will be executed with greater aggressiveness than a weaker or less certain signal. A middle ground will be to put limit orders between best current bid and offer.

ii. Large vs Small Order: a large order may be broken into many smaller orders over a window of time, but risk price moving in adverse direction. Size of chunk depends on transaction cost model estimate, and analysis of correct level of aggressiveness.

iii. Hidden vs Visible Order: a queue as a visible order gives away a bit of information. Hidden order will provide no information to the market, staving off imbalances, but reduces priority of trade in the queue. Algorithmic trading utilising hidden order is 'iceberging', which is taking a single larger order and chopping it into many smaller chunks, most posted to order book as hidden orders.

iv. Order Routing: if there are several pools of liquidity for the same instrument, smart order routing will be used, which determines which pool of liquidity is most suitable for sending a given order. Depth of liquidity on various ECNs and connectivity speeds are also considered in smart order routing.

v. Cancelling and Replacing Orders: traders may place larger number of orders with no intention of execution, then rapidly cancelling them and replacing them with other orders. This allows gaining of information on how market responds to the changing depth of the book, providing information on how to profit from the pattern of reaction. If trader wants to buy a large number of shares, he may enter a large number of small orders to sell the shares further away from market and cancel, improving market perception.

Definition 1.3.15. *High Frequency Trading*
Alpha driving strategies on extremely near-term bets (seconds or less) are *microstructure alphas*, focusing on liquidity patterns in order book. Larger quants may also use this to guide execution models, improving costs of entering trades. Small differences over a single trade add up significantly in the long run.To trade microstructure alpha as independent high frequency strategies, large investments in infrastructure and research must be done. Machine learning techniques may also be used to discern patterns in execution of other player orders. The more inferior the execution models, the easier it is to discern the pattern, allowing the ML strategy to profit from these patterns in the future. Patterns in the shorter timescale are somewhat stable.

Definition 1.3.16. *HFT Shark Strategy*
Designed to detect large orders that are iceberged, by sending series of very small trades; if each of these small orders get filled quickly, this may be a sign of a large and iceberged order. The shark simply front-run this large, hidden order by placing visible trades in front of the iceberged order. The iceberg strategy must then push prices up to execute trades. When the iceberged order is complete, prices will be pushed up favourably for the shark, which can then exit the position with a quick and relatively riskless profit.

Remark 1.3.17. *HFT Trading Infrastructure*
Using a broker that act as trading agent allows the infrastructure requirements to be handled by the broker, instead of dealing with the regulatory and other constraints.
High frequency strategies may use colocation or sponsored access. Colocation setup is where trader attempts to place trading servers as physically close to the exchange as possible.
Financial Information eXchange (FIX) protocol is the choice of real-time electronic communication among users. The software that implements the FIX protocol is free and open source (FIX engine). High frequency traders will likely build their own FIX engines to ensure optimal speeds.

1.3.6 Research

Definition 1.3.18. *Scientific Method*

1. Researcher observe a phenomenon in the market and construct a theory.
2. Researcher seeks out information to test the theory.
3. Researcher tests the theory, and with enough confidence, risk some capital on the validity of the theory.

Remark 1.3.19. *Sources of Alpha Idea Generation*

1. Observing the market, using the scientific method to test the theory
2. Academic literature, requiring significant time to read academic journals, working papers, and conference presentations for ideas. Literature from other fields such as astronomy, physics, or psychology, may provide ideas relevant to quant finance problems.
3. Migration of a researcher or portfolio manager from one quant shop to another.

4. Lessons from activities of discretionary traders

Remark 1.3.20. *Model Quality Assessment*

i. Cumulative profit graph: if profit profile is not smooth, with long periods of inactivity, sharp losses and gains, then the model may have issues
ii. Average annual rate of return: indicates how well the strategy made on historical data
iii. Variability of returns: the less variable the level of returns, the better the strategy. May look at lumpiness of returns, which is the portion of strategy's total returns that comes from periods that are significantly above average (measures consistency of returns).
iv. Worse Peak-to-Valley Drawdowns: measures maximum decline from any cumulative peak in profit curve. The lower the drawdown the better the strategy. Also, to measure recovery period after drawdowns; the shorter the recovery period the better the strategy.
v. Predictive Power: R-squared statistic may be used, which shows how much of the variability of the predicted asset have been accounted for. A exceedingly high R^2 in would be 0.05 out of sample. Instrument returns may be bucketed by deciles; a model with reliable predictive power is one that appropriately buckets the instruments correctly.
vi. Percentage Winning Trades, Winning Time Periods: whether the strategy tends to make profits from a small portion of trades that do very well, or from a large number of trades.
vii. Ratios of Returns vs Risk: Statistics such as risk-adjusted return, Sharpe ratio, information ratio, Sterling ratio, Calmer ratio, Omega ratio.
viii. Relationship with Other Strategies: value-add of new strategy compared with results of existing strategy with and without the new idea.
ix. Time decay: understand strategy returns if trades are initiated on lagged basis after receiving a trading signal. Determine strategy sensitivity to timeliness with information received, and crowdedness of strategy.
x. Sensitivity to specific parameters: high quality strategy has small changes in outcomes from slight changes in parameters. Or else this may be a sign that model may be overfitted.
xi. Overfitting: plot a graph of parameter value vs function outcome; a good model has a flatter curve with no jumps. Models that are parsimonious (less parameters) uses less assumptions, hence less overfitting.

Remark 1.3.21. *Other Considerations in Model Testing*
Overestimation of trading costs may cause portfolio to hold positions for longer than optimal, and underestimation may result in high portfolio turnover and bleed from trading costs. Assumptions on availability of short positions must also be made; hard-to-borrow lists must be taken into consideration.

1.3.7 Risk Assessment

Definition 1.3.22. *Model Risks*
Quant models has model risk, the risk that the model does not accurately describe, match, or predict the real-world phenomenon. Each component of the quant model may all have model risk.

i. Inapplicability of Modelling: occurs when quant model is mistakenly applied to a problem. May also occur with misapplication of a technique to a given problem.
ii. Model Misspecification: occurs when the model doesn't fit the real world. Model may work fine most of the time, but fail when an extreme event occurs.
iii. Implementation Errors: erors in programming or architecting systems. Architectural error may also occur when models are loaded in a wrong sequence.

Definition 1.3.23. *Regime Change Risk*
Quant models are based on relationships prevalent in historical data. If there is a regime change, the historical relationships and behaviour may be altered, hence the model may lose effectiveness.

Definition 1.3.24. *Exogenous Shock Risk*
Risks driven by information that is not internal to the market, i.e., terrorist attacks, start of wars, bank bailouts, change in regulation such as in shorting rules. May require discretionary overrides.

Definition 1.3.25. *Contagion Risk*
Happens when other investors hold the same strategies. First part of risk factor relates to how crowded the quant strategy is. Second part relates to what else is held by other investors that could force them to exit the quant strategy in a panic (ATM effect).

Quant liquidation criss may be driven by size and popularity of quantitative strategies, subpar returns from operators leading up to the crisis, the practice of funds cross-collateralising many strategies against each other, and risk targeting (risk managers target a specific level of volatility for their funds or strategies).

Method 1.3.26. *Risk Monitoring Methods*

i. Exposure Monitoring Tools: with current positions held, the positions are grouped for the various exposures (i.e., valuation, momentum level, volatility) to monitor gross and net exposure to various sectors and industries, buckets of market capitalisation, various style factors.

ii. Profit and Loss Monitors: with current portfolio, compare that with previous day closing price. Intraday performance charts are used. May also look at source of profit, hit rate (percentage of time strategy makes money on a given position).

iii. Execution Monitors: shows progress of executions, i.e., which orders are currently being worked on, which ones are completed, with transaction size and prices. Fill rates for limit orders are used for more passive execution strategies. Slippage and market impact are also monitored.

iv. System Performance Monitors: checks for software and infrastructure errors. Checks performance of CPUs, speeds of various stages of automated processes, latency in communication of messages.

1.4 Exploratory Data Analysis

1.4.1 Data Taxonomy

A brief overview of the types of data used in systematic trading.

Four essential types of financial data

Fundamental Data	Market Data	Analytics	Alternative Data
Assets	Price/Yield/IV	Analyst Recommendation	Satellite/CCTV
Liabilities	Volume	Credit Ratings	Google Searches
Sales	Dividend/Coupons	Earnings Expectations	Twitter/Chats
Costs/Earnings	Open Interest	News Sentiment	Metadata
Macro Variables	Quotes/Cancellations	. . .	. . .
. . .	Aggressor Side		
	. . .		

Remark 1.4.1. *Fundamental Data Characteristics*

i. Data published is indexed by last date included in report, which precedes date of release.
ii. Data is often backfilled or re-instated, and data vendor may overwrite initial values with corrections.
iii. Data is extremely regularised and low frequency.

Remark 1.4.2. *Market Data Characteristics*

i. Raw feed contains unstructured information, such as FIX messages (allow full construction of trading book), or full collection of BWIC (bids wanted in competition) responses.
ii. FIX data is not trivial to process, $\sim$ 10TB generated on daily basis

Remark 1.4.3. *Analytics Data Characteristics*

i. Derivative data as processed based on original source. Signal already extracted from the original source.
ii. Costly, methodology used in production may be biased or opaque.

Remark 1.4.4. *Alternative Data Characteristics*

i. Produced by individuals, business processes, and sensors.
ii. Primary information that has not made it to other sources.
iii. Cost and privacy concerns. May be useful if it annoys data infrastructure team.

Definition 1.4.5. *Reference Data*

i. Trading Universe: evolving daily to incorporate new listings, de-listings etc. Knowing when a particular stock no longer trades is important to avoid survivor bias.
ii. Symbology Mapping: ISIN, SEDOL, RIC, Bloomberg Tickers etc. Data is not static, symbols may change, complicating historical data merges. Mapping needs to persist as point-in-time data and allow for historical 'as-of-date' usage, require implementation of bi-temporal data structure.
iii. Ticker Changes: for reasons described in symbology mapping. To maintain historical table of ticker changes to seamlessly go up and down time series data.
iv. Corporate Actions Calendars: contain stock and cash dividends (announcement, execution date), stock splits, reverse splits, rights offer, mergers and acquisitions, spin off, free float or shares outstanding adjustments, quotation suspensions etc.
For dividends, announcements may coincide with more volatility, jumps in price time series. Allow building of strategies that look to benefit from the added volatility.
For stock splits, reverse splits, rights offers, all historical data need to be adjusted backward to reflect the split (both volume and price).
For M&A, spin-offs, to account for changes in valuation, hence used in Merger Arbitrage strategies.
Suspensions result in gaps in data, may impact backtesting.
v. Static Data: country, sector, primary exchange, currency, and quote factor. May be used to group instruments based on fundamental similarities (hence for pairs trading). Maintaining a table of quotation currency per instrument necessary to aggregate positions at portfolio level.
vi. Exchange Specific Data: individual exchanges have variety of differences to be accounted for when designing trading strategies. First group of information concerns the hours and dates of operations:

1. Holiday Calendar: Strategies trading simultaneously in several markets and leveraging correlation may not perform as well if one market is closed and another is open.
2. Exchange Sessions Hours: Different sessions (Pre-Market, Continuous Core, After-Hour etc.); auction times and respective cutoff times for oder submission; lunch break restricting intraday trading and auctions before/after lunch; settlement times for futures market. Daylight Saving Time (DST) adjustments; length of trading hours during course of the year; different trading hours by venue.
3. Disrupted Days: Exchange outages or trading disruptions, market data issues. To be recorded so they can be filtered out when building or testing strategies.

Second group of information governing the mechanics of trading:

1. Tick Size: Minimum eligible price increment; may vary by instrument and as a function of price.
2. Trade and Quote Lots: Minimum size increment for quotes or trades.
3. Limit-Up and Limit-Down Constraints: Maximum daily fluctuations of securities, and whether trading is paused or can only be traded at better prices than the threshold.
4. Short Sell Restrictions: Restrict shot sells not to trade at price worse than last price, or not to create a new quote that will be lower than the lowest prevailing quote. Impact ability to source liquidity.

vii. Market Data Condition Codes: vary per exchange and asset class, and each market event may be attributed to several codes at once. To build mapping table of condition codes and what they mean (i.e., auction trade, lit or dark trade, cancelled or corrected trade, regular trade, off-exchange trade reporting, block-size trade, trade originating from multi-leg order such as option spread trade etc.). To access liquidity for trading algorithm, trades published for reporting purposes must be excluded and not be used to update some of the aggregated daily data used in construction of trading strategies.

viii. Special Day Calendars: days wth distinct liquidity characteristics to be accounted for in both execution strategies and in alpha generation process. These (non-exhaustive) irregular events may be:

1. Half trading days preceding Christmas and following thanksgiving in US
2. Ramandan even in Turkey
3. Taiwan market opening on weekend to make up for lost trading days during holiday periods
4. Korean market changing trading hours on day of nationwide university entrance exam
5. Brazilian market opening late on day following the Carnival
6. Last trading days of months and quarters (investors rebalance portfolios)
7. Index rebalancing dates, where intraday volume distribution is significantly skewed toward EODs
8. Options and futures expiry dates (quarterly/monthly expiry, Triple Witching in US, Special Quotations in Japan) where excess trading volume and different intraday patterns result from hedging activity and portfolio adjustments.

Model normal days first. Special days are modelled either independently, or using normal days as baseline.

ix. Futures-Specific Reference Data: to know which contract was live at any point of time by using expiry calendar, and the most liquid contract. Equity index futures are most liquid for first contract available (front month), energy futures such as oil are more liquid for second contract. Hence to know which contract carry the most significant price formation characteristics, and what is true liquidity available.
Note there is no real standardised expiry frequency that applies across markets. When computing rolling-window metrics, to account for potential roll dates (due to investors rolling forward positions) that may have happened during the time span. May blend volume time series prior to roll date and after roll date.
Futures market also have different market phases during the day with significantly different liquidity characteristics. Various market data metrics (volume profile, average spread, average bid-ask sizes etc) should be computed separately for each market phase by maintaining a table of start and end times of each session for each contract.

x. Options-Specific Reference Data (Options Chain): expiry date and strike price combination (option chain). Map of equity tickers to option tickers with strike and expiry dates allow for design for more complex investment and hedging strategies (i.e., distance to strike, change in open interest of puts and calls).

xi. Market-Moving News Releases: macroeconomic announcements. To maintain calendar of dates and times of their occurrences to assess their impact on strategies. Central bank announcements or meeting minutes releases about major economies (FED/FOMC, ECB, BOE, BOJ, SNB), Non-Farm Payrolls, Purchasing Managers' Index, Manufacturing Index, Crude Oil Inventories etc.
Stock-specific releases such as earnings calendars, specialised sector events (for healthcare, biotech etc).

xii. Related Tickers: tickers that are related to each other as they fundamentally represent the same underlying asset. Allows efficient opportunity exploitation. Primary tickers to composite tickers mapping (for markets with fragmented liquidity), dual listed/fungible securities in US and Canada, ADR or GDR, local and foreign boards in Thailand etc.

xiii. Composite Assets: ETFs, Indexes, Mutual Funds etc. May be used to achieve desired exposures, or as cheap hedging instruments, and provide arbitrage opportunities when they deviate from NAV. To maintain information such as time series of their constituents and value of any cash component, divisor used to translate NAV into quoted price, constituent weights.

xiv. Latency tables: for higher frequency trading strategies. Contains distribution of latency between different data centres for more efficient order routing, and reordering data that are recorded in different locations.

Definition 1.4.6. *Market Data Feed*

i. Level I Data (Trade and BBO Quotes): trades and top of book quotes. Enough to reconstruct Best Bid and Offer (BBO). Also contains information in form of trade status (cancelled, reported late etc), trade and quote qualifiers (odd lot, normal trade, auction trade, Intermarket Sweep, average price reporting, on which exchange etc). May be used to analyse sequence of events and decide if a given print should be used to update the last price and total volume traded at a point in time.

ii. Level II Data (Market Depth): addition of quote depth data, displays all lit limit order book updates (price changes, addition or removal of shares quoted) at any level in the book, for all of the lit venues in fragmented markets.

iii. Level III Data (Full Order View): message data. Each order arriving is attributed a unique ID for tracking over time, and is precisely identified when it is executed, cancelled, or amended. Possible to build a full (with national depth) book at any moment intraday. Example from US market:

   1. Timestamp: number of milliseconds after midnight
   2. Ticker: equity symbol (up to 8 char)
   3. Order: Unique order ID
   4. T: message type. ’B’ is add buy order; ’S’ is add sell order’; ’E’ is execute outstanding order in part; ’C’ is cancel outstanding order in part; ’F’ is execute outstanding order in full; ’D’ is delete outstanding order in full; ’X’ is bulk volume for cross event; ’T’ is execute non-displayed order
   5. Shares: order quantity for ’B’, ’S’, ’E’, ’X’, ’C’, ’T’ messages. Zero for ’F’, ’D’ messages
   6. Price: order price for ’B’, ’S’, ’X’, ’T’ messages. Zero for cancellation and executions. Last 4 digits are decimal, padded to right with zeroes. Divide by 1000 to convert to currency value.
   7. MPID: Market Participant ID associated with transaction (4 char)
   8. MCID: Market Centre Code for originating exchange (1 char)

   A few special types of orders worth mentioning are:

   1. Order subject to price sliding: execution price may be one cent worse than display price at NASDAQ; ranked at locking price as hidden order, displayed at the price one minimum price variation inferior to locking price. New order ID will be used if order is replaced as a display order.
   2. Pegged order: based on NBBO, not routable, new timestamp given upon repricing; display rule vary over exchanges
   3. Mid-point peg order: non-displayed, may result in half-penny execution
   4. Reserve order: displayed size is ranked as displayed limit order; reserve size is behind non-displayed orders and pegged orders in priority.
      Minimum display quantity is 100, amount replenished from reserve size when it falls below 100 shares; New timestamp created, displayed size re-ranked upon replenishment.
   5. Discretionary order: displayed at one price while passively trading a more aggressive discretionary price. Order becomes active when shares are available within discretionary price range. Order ranked last in priority. Execution price may be worse than display price.
   6. Intermarket sweep order: can be executed without need for checking prevailing NBBO.

   Using these data, we may model: the pattern of inter-arrival times of various events; arrival and cancellation rates as a function of distance from nearest touch price; arrival and cancellation rates as a function of other information, such as in the queue on either side of the book, order book imbalance etc.
   Once modelled, we may analyse: the impact of market order on limit order book; chances for limit order to move up the queue from given entry position; probability of earning the spread; expected direction of price movement over a short horizon.

Definition 1.4.7. *Binned Data*

i. Open, High, Low, Close (OHLC) and Previous Close Price: indication on trading activity and intraday volatility. Distance traveled between lowest and highest points is indication of market sentiment. Previous close has to be adjusted for corporate actions and dividends.

ii. Last Trade before Close (Price/Size/Time): how much the close price may have jumped in final moments of trading; how stable it is as a reference value for next day.

iii. Volume: trading activity indicator, especially when level jumps from long term average. Collect volume breakdown between lit and dark venues for execution strategies.

iv. Auctions Volume and Price: price discovery event when significant volume prints occur.

v. VWAP: indication of trading activity on the day. Easier to algorithmically execute large orders with VWAP than a single print.

vi. Short Interest/Days-to-Cover/Utilisation: good proxy for investor position. Short pressure an indication of upcoming short term moves: large short interest is bearish view from institutional investors. Utilisation level of available securities to borrow gives indication of how much room is left for future shorting. Days-to-Cover to assess magnitude of potential short squeeze (if sellers unwind position, fraction of available daily liquidity needed); larger value indicates larger potential of sudden upswing on heavily shorted securities.

vii. Futures Data: insight into activity or large investors through open interest data. Offer arbitrage opportunities if their basis exhibits mis-pricing compared to one's dividend estimates.

viii. Index-Level Data: source of relative measures for instrument specific features (index OHLC, volatility). Normalised features identify individual instruments deviating from their benchmarks.

ix. Options Data: information on position of traders through open interest and Greeks.

x. Asset Class Specific: yield/benchmark rates (repo, 2y, 10y, 30y), CDS spreads, US Dollar Index

Definition 1.4.8. *Granular Intraday Microstructure Activity*

i. Number and Frequency of Trades: proxy for activity level, and how continuous it is. Low number of trades mean harder execution, and may be more volatile

ii. Number and Frequency of Quote Updates: similar proxy for activity level

iii. Top of Book Size; proxy for liquidity of instruments (larger top of book size makes it possible to trade larger order quasi immediately)

iv. Depth of Book (price and size): similar proxy for liquidity

v. Spread Size (average, median, time weighted average): proxy for cost of trading. Parametrised distribution used to identify opportunities if they are cheap or expensive

vi. Trade size (average, median): to identify intraday liquidity opportunities when examining volume available in the order book.

vii. Ticking time (average, median): representation of how often one should expect changes in the order book first level. For execution algorithms for which the frequency of updates (adding/cancelling child orders, re-evaluating decisions etc.) should commensurate with characteristics of the traded instrument.

Daily distribution can be used as start of day estimates and updated intraday with online Bayesian updates. Last group of daily data is derived from previous two groups but stored pre-computed to save time during research phase, or to be used as normalising values:

i. X-day Average Daily Volume (ADV) / Average Auction Volume

ii. X-day Volatility (close-to-close, open-to-close etc)

iii. Beta with respect to index or sector (plain beta, or asymmetric up-days/down-days beta)

iv. Correlation matrix

When binning data, this may be grouped into bins ranging from a few seconds to 30 minutes. Minute bar data are used for volume and spread profiles to prevent introducing excess noise due to market friction.

Definition 1.4.9. *Fundamental Data and Other Data*

i. Key Ratios: Earnings Per Share (EPS), Price-to-Earning (P/E), Price-to-Book (P/B), etc.

ii. Analyst Recommendations: aggregated values given consensus valuation

iii. Earnings data: estimations by research analysts provide quarterly earning estimates which can be used as indication of performance of stock before actual value is published

iv. Holders: sudden changes in ownership indicate changes in sentiment by sophisticated investor

v. Insiders Purchase/Sale: indicator of future stock price moves from group of people who have access to best possible information about the company

vi. Credit Ratings: credit downgrades resulting in higher funding costs have negative impact on equity prices

1.4.2 Financial Data Structures

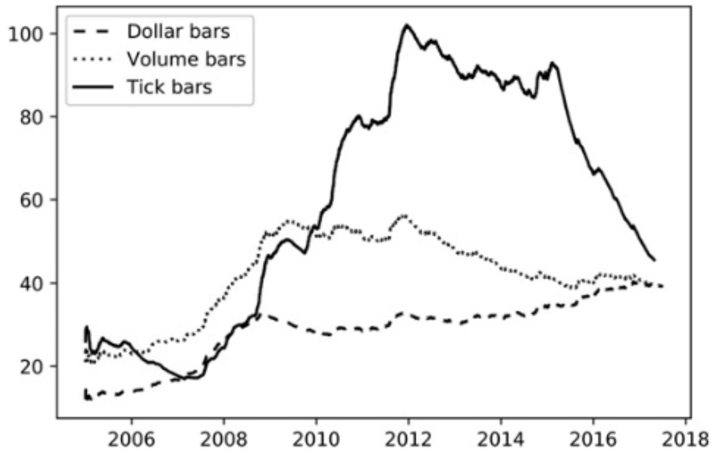

Figure 2: Average daily frequency of tick, volume, and dollar bars

Remark 1.4.10. *Standard BARS*
Method to transform a series of observations arriving at irregular frequency into a homogeneous series derived from regular sampling.

i. Time Bars: obtained by sampling information at fixed intervals. Information collected includes timestamp, volume-weighted average price (VWAP), open price, close price, high price, low price, volume etc.
To be avoided as markets do not process information at constant time interval. Time bars oversample information in low-activity periods and under-sample information in high-activity periods. Time bars exhibit poor statistical properties, i.e., serial correlation, heteroscedasticity, non-normality of returns.

ii. Tick Bars: sample variables extracted each time a pre-defined number of transactions take place. Allows synchronisation of sampling with a proxy of information arrival.
Sampling as a function of trading activity creates returns closer to IID Normal (Thierry and Helyette (2000)). When constructing tick bars, to be aware of outliers, as many exchanges carry out auction at open and at close; order book accumulates bids and offers without matching. Order fragmentation introduces some arbitrariness in number of ticks. Matching engine protocols may split one fill into multiple artificial partial fills as a matter of operational convenience.

iii. Volume Bars: samples every time a pre-defined amount of security's units that have been exchanged. Achieves better statistical properties than sampling tick bars.
Convenient artefact for studying market microstructure theories.

iv. Dollar Bars: samples an observation every time a pre-defined market value is exchanged. Used when the analysis involves significant price fluctuations. Robust against corporate actions such as splits, reverse splits, issuance of new shares, buying back existing shares.
Bar size could be dynamically adjusted as a function of free-floating market cap of a company or outstanding amount of issued debt.

Remark 1.4.11. *Information-Driven Bars*
Method to sample more frequently when new (micro-structural) information arrives to the market.

i. Tick Imbalance Bars: sample bars whenever tick imbalance exceeds expectations. To determine tick index T such that accumulation of signed ticks exceeds a given threshold.
Let $\{(p_t, v_t)\}_{t=1,\dots,T}$ be sequence of ticks where p_t and v_t is the price and volume associated with tick t.
Let tick rule define a sequence $\{b_t\}_{t=1,\dots,T}$ where

$$b_t = \begin{cases} b_{t-1} & \text{if } \Delta p_t = 0 \\ \frac{|\Delta p_t|}{\Delta p_t} & \text{if } \Delta p_t \neq 0 \end{cases}$$

The tick imbalance at time T is defined as

$$\theta_T = \sum_{t=1}^{T} b_t$$

Compute expected value of θ_T at beginning of the bar,

$$E_0[\theta_T] = E_0[T](P[b_t = 1] - P[b_t = -1]) = E_0[T](2P[b_t = 1] - 1)$$

where $E_0[T]$ is expected size of tick bar, $P[b_t = 1]$ and $P[b_t = -1]$ is unconditional probability that a tick is classified as a buy and sell. In practice, $E_0[T]$ and $(2P[b_t = 1]-1)$ may be estimated as an exponentially weighted moving average of T and b_t values from prior bars.
Define the tick imbalance bar (TIB) as a T^* contiguous subset of ticks such that

$$T^* = \arg\min_T\{|\theta_T| \geq E_0[T] \mid 2P[b_t = 1] - 1\}$$

where the size of expected imbalance is implied by $|2P[b_t = 1] - 1|$.
When θ_T is more imbalanced than expected, a low T will satisfy the conditions.
TIBs are produced more frequently under presence of informed trading (asymmetric information that triggers one-side trading). TIBs are buckets of trades containing equal amounts of information.

ii. Volume/Dollar Imbalance Bars: sample bars when volume or dollar imbalances diverge from expectations. First, define imbalance at time T as

$$\theta_T = \sum_{t=1}^{T} b_t v_t$$

where v_t may represent ether number of securities traded (VIB) or dollar amount traded (DIB).
The expected value of θ_T at the beginning of the bar is then computed as

$$\begin{aligned} E_0[\theta_T] &= E_0\left[\sum_{t|b_t=1}^{T} v_t\right] - E_0\left[\sum_{t|b_t=-1}^{T} v_t\right] \\ &= E_0[T](P[b_t = 1]E_0[v_t|b_t = 1] - P[b_t = -1]E_0[v_t|b_t = -1]) \\ &= E_0[T](v^+ - v^-) \end{aligned}$$

where the initial expectation of v_t is decomposed into component contributed by buys and sells. Then

$$E_0[\theta_T] = E_0[T](2v^+ - E_0[v_t])$$

In practice, $E_0[T]$ and $(2v^+ - E_0[v_t])$ may be estimated as exponentially weighted moving average of T and $b_t v_t$ values from prior bars. Next, define VIB or DIB as a T^*-contiguous subset of ticks such that

$$T^* = \arg\min_T\{|\theta_T| \geq E_0[T] \mid 2v^+ - E_0[v_t]\}$$

where the size of expected imbalance is implied by $|2v^+ - E_0[v_t]|$.
When θ_T is more imbalanced then expected, a low T will satisfy the conditions.
VIB and DIB addresses concerns on tick fragmentation and outliers, and also addresses the issues of corporate actions, as the bar size is adjusted dynamically.

iii. Tick Runs Bars: sample bars when the sequence of buys in overall volume diverges from expectations. For the case when large traders sweep order book, use iceberg orders, or slice parent orders into multiple children, all leaving a trace of runs in the $\{b_t\}_{t=1,\ldots,T}$ sequence. Define length of current run as

$$\theta_T = \max\left\{\sum_{t|b_t=1}^{T} b_t - \sum_{t|b_t=-1}^{T} b_t\right\}$$

The expected value of θ_T at beginning of bar is computed as

$$E_0[\theta_T] = E_0[T]\max\{P[b_t = 1], 1 - P[b_t = 1]\}$$

In practice, $E_0[T]$ and $P[b_t = 1]$ may be estimated as exponentially weighted moving average of T and proportion of buy ticks from prior bars. Next, define TRB as T^*-contiguous subset of ticks such that

$$T^* = \arg\min_T \{\theta_T \geq E_0[T] \max\{P[b_t = 1], 1 - P[b_t = 1]\}\}$$

where the expected count of ticks from runs is implied by $\max\{P[b_t = 1], 1 - P[b_t = 1]\}$.
When θ_T exhibits more runs than expected, a low T will satisfy these conditions.
Instead of measuring length of longest sequence, count number of ticks of each side without offsetting.

iv. Volume/Dollar Runs Bars: sample bars when volume or dollars traded by one side exceed expectation for a bar. First, define volume or dollars associated with a run as

$$\theta_T = \max\left\{\sum_{t|b_t=1}^{T} b_t v_t - \sum_{t|b_t=-1}^{T} b_t v_t\right\}$$

where v_t may either represent volume (VRB) or dollar amount exchanged (DRB). The expected value of θ_T at beginning of the bar is then

$$E_0[\theta_T] = E_0[T] \max\{P[b_t = 1] E_0[v_t | b_t = 1], (1 - P[b_t = 1]) E_0[v_t | b_t = -1]\}$$

In practice, $E_0[T], P[b_t = 1], E_0[v_t|b_t = 1], E_0[v_t|b_t = -1]$ may be estimated as exponentially weighted moving average of T, proportion of buy ticks, buy volumes, and sell volumes from prior bars. Next, define a volume runs bar (VR) as T^*-contiguous subset of ticks such that

$$T^* = \arg\min_T \{\theta_T \geq E_0[T] \max\{P[b_t = 1] E_0[v_t|b_t = 1], (1 - P[b_t = 1]) E_0[v_t|b_t = -1]\}\}$$

expected volume from runs is implied by $\max\{P[b_t = 1] E_0[v_t|b_t = 1], (1 - P[b_t = 1]) E_0[v_t|b_t = -1]\}\}$.
When θ_T exhibits more runs than expected, volume from runs is greater than expected, a low T will satisfy these conditions.

Definition 1.4.12. *Multi-Product Series: ETF Trick*
To model a basket of securities as if it was a single cash product. To transform any complex multi-product dataset into a single dataset that resembles a total return ETF.

Method 1.4.13. *ETF Trick*
Produce a time series that reflects the value of \$1 invested. Changes in the series will reflect changes in PnL, series will be strictly positive, and implementation shortfall will be taken into account. The bars contain:

i. Raw open price of instrument $i = 1, \ldots, I$ at bar $t = 1, \ldots, T$: $o_{i,t}$

ii. Raw close price of instrument $i = 1, \ldots, I$ at bar $t = 1, \ldots, T$: $p_{i,t}$

iii. USD value of one point of instrument $i = 1, \ldots, I$ at bar $t = 1, \ldots, T$: $\varphi_{i,t}$. This includes forex rate.

iv. Volume of instrument $i = 1, \ldots, I$ at bar $t = 1, \ldots, T$: $v_{i,t}$

v. Carry, dividend, or coupon paid by instrument i at bar t: $d_{i,t}$. Variable can also be used to charge margin costs or costs of funding.

All instruments $i = 1, \ldots, I$ were tradable at bar $t = 1, \ldots, T$. Even if some instruments were not tradable over entirety of time interval $[t - 1, t]$, at least they were tradable at times $t - 1$ and t.
For basket of securities with allocation vector ω_t rebalanced (or rolled) on bars $B \subseteq \{1, \ldots, T\}$, the \$1 investment value $\{K_t\}$ is derived as

$$h_{i,t} = \begin{cases} \frac{\omega_{i,t} K_t}{o_{i,i+1} \varphi_{i,t} \sum_{i=1}^{I} |\omega_{i,t}|} & \text{if } t \in B \\ h_{i,t-1} & \text{otherwise} \end{cases}$$

$$\delta_{i,t} = \begin{cases} p_{i,t} - o_{i,t} & \text{if } (t-1) \in B \\ \Delta p_{i,t} & \text{otherwise} \end{cases}$$

$$K_t = K_{t-1} + \sum_{i=1}^{I} h_{i,t-1} \varphi_{i,t} (\delta_{i,t} + d_{i,t})$$

where $K_0 = 1$ is the initial AUM. Variable $h_{i,t}$ is the holdings of instrument i at time t, $\delta_{i,t}$ is change of market value between $t - 1$ and t for instrument i. Note profits or losses are being reinvested whenever $t \in B$, hence preventing negative prices. Dividends $d_{i,t}$ are already embedded in K_t.

The purpose of $\omega_{i,t}\left(\sum_{i=1}^{I}|\omega_{i,t}|\right)^{-1}$ is to de-lever the allocations.
Let τ_i be transaction cost associated with trading \$1 of the instrument. Three additional variables that the strategy needs to know for every observed bar t are:

i. Rebalance Costs: variable cost $\{c_t\}$ associated with allocation rebalance is

$$c_t = \sum_{i=1}^{I}(|h_{i,t-1}|\,p_{i,t} + |h_{i,t}|\,o_{i,t+1})\varphi_{i,t}\tau_i \quad \forall t \in B$$

Note c_t is not embedded in K_t, as shorting will generate fictitious proceeds when allocation is rebalanced. In code, $\{c_t\}$ is treated as a (negative) dividend.

ii. Bid-Ask Spread: the cost $\{\tilde{c}_t\}$ of buying or selling one unit of this ETF,

$$\tilde{c}_t = \sum_{i=1}^{I}|h_{i,t-1}|\,p_{i,t}\varphi_{i,t}\tau_i$$

When a unit is bought or sold, strategy must charge this cost $\tilde{c}_t$.

iii. Volume: volume traded $\{v_t\}$ is determined by least active member in the basket. Let $v_{i,t}$ be volume traded by instrument i over bar t. The number of tradable basket units is

$$v_t = \min_i\left\{\frac{v_{i,t}}{|h_{i,t-1}|}\right\}$$

Transaction costs functions may not be linear, and can be simulated by the strategy.

Method 1.4.14. *ETF Trick: Computation of Allocation Vector with PCA*
Consider an IID multivariate Gaussian process with means vector μ of size $N \times 1$, and covariance matrix V of size $N \times N$. First, perform spectral decomposition $VW = W\Lambda$, where columns in W are reordered so that elements of Λ diagonal are sorted in descending order. Second, given allocations vector ω, portfolio risk is

$$\sigma^2 = \omega' V \omega = \omega' W \Lambda W' \omega = \beta' \Lambda \beta = (\Lambda^{1/2}\beta)'(\Lambda^{1/2}\beta)'$$

where β is projection of ω on orthogonal basis. Third, Λ is a diagonal matrix, thus

$$\sigma^2 = \sum_{n=1}^{N}\beta_n^2\Lambda_{n,n}$$

The risk attributed to the nth component is

$$R_n = \beta_n^2\Lambda_{n,n}\sigma^{-2} = [W'n]_n^2\Lambda_{n,n}\sigma^{-2}$$

with $R'1_N = 1$, and 1_N is a vector of N ones.
Note $\{R_n\}_{n=1,\ldots,N}$ is distribution of risks across orthogonal components.
Next, compute vector ω which delivers user-defined risk distribution R. Note from earlier,

$$\beta = \left\{\sigma\sqrt{\frac{R_n}{\Lambda_{n,n}}}\right\}_{n=1,\ldots,N}$$

which represents allocation in new (orthogonal basis).
The allocation in old basis is $\omega = W\beta$. Rescaling ω re-scales σ, hence keeping risk distribution constant.

Method 1.4.15. *ETF Trick: Single Futures Roll*
To work with non-negative rolled series, derive price series of \$1 investment as follows:

i. Compute time series of rolled futures prices

ii. Compute return r as rolled price change divided by previous roll price

iii. Form a price series using these returns

These methods allow us to produce a continuous, homogeneous, and structured dataset from collection of unstructured financial data. Note however, that several ML algorithms do not scale well with sample size. ML algorithms achieve higher accuracy when they attempt to learn from relevant examples.

Method 1.4.16. *Sampling for Reduction*
To reduce the amount of data used to fit ML algorithm, downsampling could be used.

i. Sequential sampling at constant step size (linspace sampling)
ii. Sampling randomly using uniform distribution (uniform sampling)

Note both samples do not necessarily contain subset of most relevant observations.

Method 1.4.17. *Event-Based Sampling: CUMSUM Filter*
Bets are often placed after some event takes place, hence to let ML algorithm learn whether there is an accurate prediction function under these circumstances, CUSUM filter could be used.
This is a quality-control method, to detect shift in mean value of measured quantity away from a target value.
Let $\{y_t\}_{t=1,\dots,T}$ be IID observations arising from a locally stationary process. The cumulative sums are

$$S_t = \max\{0, S_{t-1} + y_t - E_{t-1}[y_t]\}, \quad S_0 = 0$$

An action will be recommended at the first t satisfying $S_t \geq h$ for some threshold h (filter size).
Note $S_t = 0$ whenever $y_t = E_{t-1}[y_t] - S_{t-1}$, The zero floor means some downward deviations will be skipped.
The filter is set up to identify a sequence of upside divergences from any reset level zero.
The threshold is activated when

$$S_t \geq h \Leftrightarrow \exists \tau \in [1, t] \mid \sum_{i=\tau}^{t} (y_i - E_{i-1}[y_t]) \geq h$$

This concept of run-ups can be extended to include run-downs, giving symmetric CUSUM filter.

$$\begin{aligned} S_t^+ &= \max\{0, S_{t-1}^+ + y_t - E_{t-1}[y_t]\}, \quad S_0^+ = 0 \\ S_t^- &= \min\{0, S_{t-1}^- + y_t - E_{t-1}[y_t]\}, \quad S_0^- = 0 \\ S_t &= \max\{S_t^+, -S_t^-\} \end{aligned}$$

1.4.3 Data Labelling Techniques

Method 1.4.18. *Labelling with Fixed-Time Horizon Method*
Given features matrix X with I rows, $\{X_i\}_{i=1,\dots,I}$ drawn from some bards with index $t = 1, \dots, T$, where $I \leq T$, let an observation X_i be assigned a label $y_i \in \{-1, 0, 1\}$,

$$\begin{aligned} y_i &= \begin{cases} -1 & \text{if } r_{t_{i,0}, t_{i,0}+h} < -\tau \\ 0 & \text{if } \left| r_{t_{i,0}, t_{i,0}+h} \right| \leq \tau \\ 1 & \text{if } r_{t_{i,0}, t_{i,0}+h} > \tau \end{cases} \\ r_{t_{i,0}, t_{i,0}+h} &= \frac{p_{t_{i,0}+h}}{p_{t_{i,0}}} - 1 \end{aligned}$$

where τ is a pre-defined constant threshold, $t_{i,0}$ is index of bar immediately after X_i takes place, $t_{i,0} + h$ is index of h-th bar after $t_{i,0}$, and $r_{t_{i,0}, t_{i,0}+h}$ is price return over bar horizon h.

Remark 1.4.19. *Limitations of Fixed-Time Horizon Method*

i. Time bars do not exhibit good statistical properties (as seen earlier)
ii. The same threshold τ is applied regardless of observed volatility.
Compute daily volatility at intraday estimation points, applying span of n days t an exponentially weighted moving standard deviation.

Method 1.4.20. *Labelling with Triple-Barrier Method*
Labels an observation according to first barrier touched out of three barriers.

i. Set two horizontal barriers and one vertical barrier. Horizontal barriers are defined by profit-taking and stop-loss limits, which are a dynamic function of estimated volatility (realised or implied). Third barrier is the number of bars elapsed since the position was taken (expiration limit).
ii. If upper barrier is touched first, label observation as 1. If lower barrier is touched first, label observation as -1. If vertical barrier is touched first, either label by sign of the return or with 0.

Note that the method is path-dependent. To label an observation, need to account for entire path spanning $[t_{i,0}, t_{i,0} + h]$ where h defines the vertical barrier (expiration limit). Let $t_{i,1}$ be the time of first barrier touch with return as $r_{t_{i,0}, t_{i,1}}$. The horizontal barriers may not be symmetric.

Remark 1.4.21. *Triple-Barrier Method Configurations*
Denote a barrier configuration by triplet $[pt, sl, t1]$ which are the upper barrier, lower barrier, physical barrier. Set value as 0 if barrier is inactive, and 1 if barrier is active.
The three useful configurations are:

i. $[1, 1, 1]$: to realise profit, but have set a maximum tolerance for losses and a holding period.
ii. $[0, 1, 1]$: to exit after a number of bars, unless stopped-out.
iii. $[1, 1, 0]$: take profit as long as not stopped-out.

The three less realistic configurations are:

i. $[0, 0, 1]$: equivalent to fixed-time horizon method.
ii. $[1, 0, 1]$: position held until a profit is made or maximum holding period is exceeded, without regard for immediate unrealised losses
iii. $[1, 0, 0]$: position is held until a profit is made. Could lock in loose position for years.

The two illogical configurations are:

i. $[0, 1, 0]$: aimless. Hold position until stopped-out.
ii. $[0, 0, 0]$: no barriers. Position locked forever, no label generated.

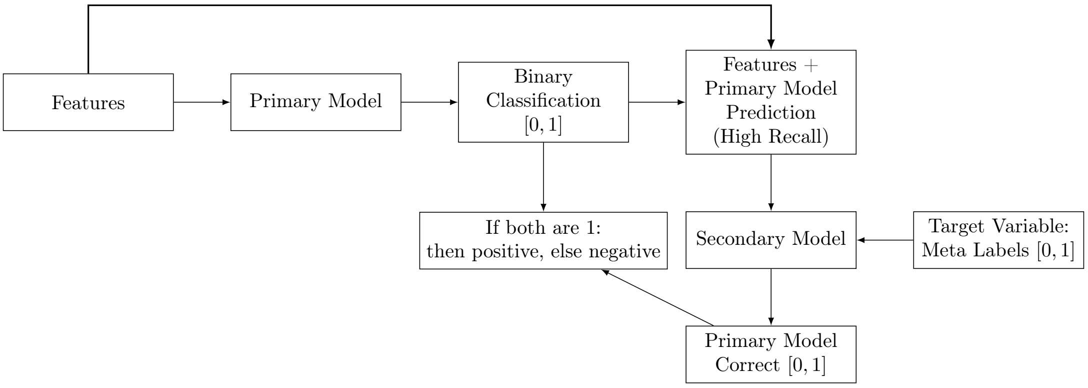
Features
Primary Model
Binary Classification [0, 1]
Features + Primary Model Prediction (High Recall)
If both are 1: then positive, else negative
Secondary Model
Target Variable: Meta Labels [0, 1]
Primary Model Correct [0, 1]

Figure 3: Meta-Labelling Process

Method 1.4.22. *Meta-Labelling*
The technique is particularly helpful to achieve higher F1-scores.
First, build a model that achieves high recall, even if precision is not particularly high. Second, correct for low precision by applying meta-labelling to positives predicted by primary model.
Meta-labelling will filter out false positives, where majority of positives have been identified by primary model.
The second model's purpose is to determine if the positive from primary model is true or false.

i. Train a primary model (binary classification)
ii. A threshold level is determined at which the primary model has a high recall, ROC curves could be used to help determine a good level.
iii. Typical features of second model are as follows:
    i. Primary model features concatenated with predictions from first model.
    ii. Market state
    iii. Features indicative of false positives
    iv. Distribution related
    v. Recent model performance

    Meta Labels are used as target variable in second model. Fit the second model
iv. Prediction from the secondary model is combined with the prediction from the primary model and only where both are true, is your final prediction true.

Remark 1.4.23. *Limitations of Meta-Labelling*

i. If model has overfit the data, meta-labelling will not add much value

ii. If every trade is not treated as an independent observation, the meta-model is forced to determine day-to-day exposures, which is the wrong way to apply the technique

iii. Technique trades recall for precision. Require a large number of trades to train on, while being happy with reduction in trade frequency

1.4.4 Data Sample Weights

Note that most of ML literature is based on IID assumption, and ML applications usually fail in finance as these assumptions are unrealistic in the case of financial time series.

Remark 1.4.24. *Overlapping Outcomes*
Let label y_i be assigned to an observed feature X_i, where $y_i = f([t_{i,0}, t_{i,1}])$ is a function over the interval. When $t_{i,1} > t_{j,0}$ and $i < j$, then y_j will depend on common return $r_{t_{j,0},\min\{t_{i,1},t_{j,1}\}}$ (over interval $[t_{j,0}, \min\{t_{i,1}, t_{j,1}\}]$). The series of labels $\{y_i\}_{i-1,\dots,J}$ are not IID whenever there is overlap between any two consecutive outcomes, i.e., $\exists i \mid t_{i,1} > t_{i+1,0}$. If this is resolved by restricting bet horizon to $t_{i,1} \leq t_{i+1,0}$, there is no overlap, but this will lead to coarse models where features sampling frequency is limited by horizon used to determine outcome. To investigate outcomes that lasted a different duration, samples have to be resampled with different frequency. In addition, if path-dependent labelling technique is to be applied, the sampling frequency will be subordinated to first barrier's touch. Hence, to use $t_{i,1} > t_{i+1,0}$, leading to overlapping outcomes.

Method 1.4.25. *Estimating Uniqueness of Label*
Let two labels y_i and y_j be concurrent at time t, both a function of at least one common return $r_{t-1,t} = \frac{p_t}{p_{t-1}} = 1$. To compute the number of labels that are a function of given return $r_{t-1,t}$:

i. For each $t = 1, \dots, T$, form a binary array $\{1_{t,i}\}_{i=1,\dots,I}$ where $1_{t,i} \in \{0, 1\}$.
Variable $1_{t,i} = 1$ if and only if $[t_{i,0}, t_{i,1}]$ overlaps with $[t-1, t]$ and $1_{t,i} = 0$ otherwise.

ii. Compute the number of labels concurrent at t, $c_t = \sum\limits_{i=1}^{I} 1_{t,i}$

Method 1.4.26. *Average Uniqueness of Label*
To estimate label's uniqueness (non-overlap) across its lifespan.

i. Uniqueness of label i at time t is $u_{t,i} = 1_{t,i} c_t^{-1}$.

ii. Average uniqueness of label i is average $u_{t,i}$ over label's lifespan, $\overline{u}_i = (\sum_{t=1}^{T} u_{t,i})(\sum_{t=1}^{T} 1_{t,i})^{-1}$.

Note that $\{\overline{u}_i\}_{i=1,\dots,I}$ are not used for forecasting the label, hence there is no information leakage.

Remark 1.4.27. *IID and Oversampling*
Probability of not selecting item i after I draws with replacement on set of I items is $(1 - I^{-1})^I$. As $I \to \infty$, note that $(1 - I^{-1})^I \to e^{-1}$. Number of unique observations drawn to be expected is $(1 - e^{-1}) \approx \frac{2}{3}$.
If maximum number of overlapping outcomes is $K \leq I$, probability of not selecting a particular item i after I draws with replacement on set of I items is $(1-K^{-1})^I$. As sample size increase, probability can be approximated as $(1 - I^{-1})^{I\frac{K}{I}} \approx e^{-\frac{K}{I}}$. Implication is that incorrectly assuming IID draws lead to oversampling.

Method 1.4.28. *Sampling with Bootstrap, Redundancy*
Sampling with bootstrapping on observations where $I^{-1} \sum_{i=1}^{I} \overline{u}_i \ll 1$, in-bag observations will increasingly be redundant to each other, and very similar to out-of-bag observations. Two solutions may be:

i. Drop overlapping outcomes before performing bootstrap.
As overlaps are not perfect, dropping an observation due to overlap will lead to extreme loss in information.

ii. Utilise the average uniqueness $I^{-1} \sum_{i=1}^{I} \overline{u}_i$ to reduce undue influence of outcomes that contain redundant information. Ensure in-bag observations are not sampled at frequency much higher than uniqueness.

Method 1.4.29. *Sequential Bootstrap*
Draws made according to changing probability that controls for redundancy.

i. Observation X_i is drawn from uniform distribution, $i \sim U[1, I]$.
Probability of drawing any value i is $\delta_i^{(1)} = I^{-1}$.

ii. Second draw, to reduce probability of drawing observation X_j with highly overlapping outcome.
Let φ be sequence of draws (may include repetitions), where $\{\varphi^{(1)}\} = \{i\}$.
Uniqueness of j at time t is $u_{t,j}^{(2)} = 1_{t,j}(1 + \sum_{k \in \varphi^{(1)}} 1_{t,k})^{-1}$, which is the uniqueness from adding alternative j's to existing sequence of draws $\varphi^{(1)}$.

Average uniqueness of j is average $u_{t,j}^{(2)}$ over j's lifespan, $\overline{u}_j^{(2)} = (\sum_{t=1}^T u_{t,j})(\sum_{t=1}^T 1_{t,j})^{-1}$.
A second draw can be made based on updated probabilities $\{\delta_j^{(2)}\}_{j=1,\ldots,I}$:

$$\delta_j^{(2)} = \overline{u}_j^{(2)} \left(\sum_{k=1}^I \overline{u}_j^{(2)} \right)^{-1}$$

where $\sum_{j=1}^I \delta_j^{(2)} = 1$. Do a second draw, update $\varphi^{(2)}$, and re-evaluate $\{\delta_j^{(3)}\}_{j=1,\ldots,I}$.

iii. Process is repeated until I draws have taken place.

Process draws samples much close to IID, verified by increase in $I^{-1}\sum_{i=1}^I \overline{u}_i$.

Method 1.4.30. *Weighting Observations by Uniqueness and Absolute Return*
Let labels be a function for return sign ($\{-1,1\}$ for standard label, $\{0,1\}$ for meta-label). The sample weights can be defined in terms of sum of attributed returns over event's life-span, $[t_{i,0}, t_{i,1}]$,

$$\tilde{w}_i = \left| \sum_{t=t_{i,0}}^{t_{i,1}} \frac{r_{t-1,t}}{c_t} \right|, \quad w_t = \tilde{w}_i \left(\sum_{j=1}^I \tilde{w}_j \right)^{-1}$$

where $\sum_{i=1}^I w_i = I$. The method weigh an observation as a function of absolute log returns that can be attributed uniquely to it. Lower returns should be assigned higher weights.

Method 1.4.31. *Time Decay Weighting*
To let sample weights decay as new observations arrive.
Let $d[x] \geq 0 \ \forall x \in [0, \sum_{i=0}^I \overline{u}_i]$ be time-decay factors multiplying sample weights from earlier.
The final weight has no decay, $d[\sum_{i=1}^I \overline{u}_i] = 1$, and all other weights will adjust relative to that.
Let $c \in (-1, 1]$ be user-defined parameters that determines decay function as follows:

i. If $c \in [0,1]$, then $d[1] = c$ with linear decay

ii. If $c \in (-1, 0)$, then $d[-c\sum_{i=1}^I \overline{u}_i] = 0$, with linear decay between $[-c\sum_{i=1}^I \overline{u}_i, \sum_{i=1}^I \overline{u}_i]$, and $d[x] \ \forall x \leq -c\sum_{i=1}^I \overline{u}_i$.

If given linear piecewise function $d = \max\{0, a + bx\}$, requirements are met by following boundary conditions:

i. $d = a + b\sum_{i=1}^I \overline{u}_i = 1 \Rightarrow a = 1 - b\sum_{i=1}^I \overline{u}_i$

ii. Contingent on c:

1. $d = a + b0 = c \Rightarrow b = (1-c)(\sum_{i=1}^I \overline{u}_i)^{-1} \quad \forall c \in [0,1]$
2. $d = a - bc\sum_{i=1}^I \overline{u}_i = 0 \Rightarrow b = [(c+1)\sum_{i=1}^I \overline{u}_i]^{-1} \ \forall c \in (-1, 0)$

In the implementation, decay takes place according to cumulative uniqueness. Note that

i. $c = 1$ means there is no time decay

ii. $0 < c < 1$ means weights decay linearly over time, but every observation still receives strictly positive weight, regardless of age

iii. $c = 0$ means weights converge linearly to zero over time

iv. $c < 0$ means oldest portion cT of observations receive zero weight (erased from memory)

Method 1.4.32. *Class Weighting*
Weights for underrepresented labels. Critical in classification problems where the most important classes have rare occurrences. To assign higher weights to samples associated with those rare labels.

1.4.5 Fractionally Differentiated Features

Standard stationarity transformations (i.e. integer differentiation) reduce signal by removing memory. Although stationarity is necessary for inferential purposes, it is rarely the case that we want all memory to be erased. Fractionally differentiated processes exhibit long-term persistence and anti-persistence, hence enhancing the forecasting power compared to standard ARIMA approach.

Definition 1.4.33. *BackShift Operator*
Let B be the backshift operator applied to a matrix of real-valued features $\{X_t\}$, where $B^k X_t = X_{t-k}$ for any integer $k \geq 0$. By binomial expansion, we then have

$$\begin{aligned}(1-B)^d = \sum_{k=0}^{\infty} \binom{d}{k} (-B)^k &= \sum_{k=0}^{\infty} \prod_{i=0}^{k-1} (d-i) \frac{(-B)^k}{k!} = \sum_{k=0}^{\infty} (-B)^k \prod_{i=0}^{k-1} \frac{d-i}{k-i} \\ &= 1 - dB + \frac{d(d-1)}{2!} B^2 - \frac{d(d-1)(d-2)}{3!} B^3 + \cdots\end{aligned}$$

Remark 1.4.34. *Properties of Fractionally Differentiated Features*
Let d be a real (non-integer) positive number. The arithmetic series consists of dot product

$$\begin{aligned}\tilde{X}_t &= \sum_{k=0}^{\infty} \omega_k X_{t-k} \\ \omega &= \left\{1, -d, \frac{d(d-1)}{2!}, -\frac{d(d-1)(d-2)}{3!}, \ldots, (-1)^k \prod_{i=0}^{k-1} \frac{d-1}{k!}, \ldots\right\} \\ X &= \{X_t, X_{t-1}, \ldots, X_{t-k}, \ldots\}\end{aligned}$$

where ω are the weights, X are the values. Properties of these features are:

i. Long memory: if d is a positive integer number, then

$$\prod_{i=0}^{k-1} \frac{d-i}{k!} = 0 \quad \forall > d$$

and memory beyond that point is cancelled.

ii. Iterative weight generation: given sequence of weights ω, for $k = 0, \ldots, \infty$, the weights are

$$\omega_k = -\omega_{k-1} \frac{d-k+1}{k}, \quad \omega_0 = 1$$

iii. Convergence: For $k > d$, if $\omega_{k-1} \neq 0$, then

$$\left|\frac{\omega_k}{\omega_{k-1}}\right| = \left|\frac{d-k+1}{k}\right| < 1$$

and $\omega_k = 0$ otherwise. Hence weights converge asymptotically to zero.
For positive d and $k < d+1$, then $\frac{d-k+1}{k} \geq 0$, which makes initial weights alternate in sign.
For non-integer d, once $k \geq d+1$, ω_k will be negative if int$[d]$ is even, and positive otherwise.
In summary, $\lim_{k\to\infty} = 0^-$ when int$[d]$ is even, and $\lim_{k\to\infty} = 0^+$ when int$[d]$ is odd.
In special case $d \in (0,1)$, that $-1 < \omega_k < 0 \; \forall k > 0$. Alternate weight signs makes $\{\tilde{X}_t\}_{t=1,\ldots,T}$ stationary, as memory wanes or is offset over the long run.

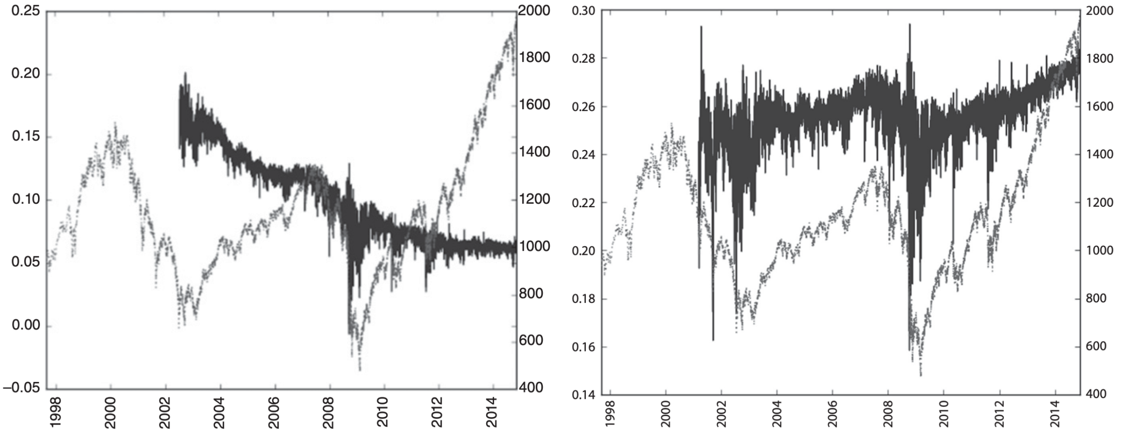

Figure 4: Fractional differentiation controlling for weight loss with expanding and fixed-width window

Method 1.4.35. *Expanding Window*
Given time series T with real observations $\{X_t\}_{t=1,\ldots,T}$, for each l, the relative weight loss is defined as

$$\lambda_l = \sum_{j=T-l}^{T} |\omega_j| \bigg/ \sum_{i=0}^{T-1} |\omega_i|$$

Given tolerance level $\tau \in [0,1]$, determine value l^* such that $\lambda_{l^*} \leq \tau$ and $\lambda_{l^*+1} > \tau$. This value l^* corresponds to the first results $\{\tilde{X}_t\}_{t=1,\ldots,l^*}$, where weight-loss is beyond acceptable threshold $\lambda_t > \tau$.
From Remark 1.4.34, it is clear λ_{l^*} depends on convergence speed of $\{\omega_k\}$, which in turn depends on $d \in [0,1]$. For $d = 1, \omega_k = 0 \ \forall k > 1$, and $\lambda_l = 0 \ \forall l > 1$, hence it suffices to drop $\tilde{X}_1$.
As $d \to 0^+$, l^* increases, and larger portion of initial $\{\tilde{X}_t\}_{t=1,\ldots,l^*}$ needs to be dropped to keep the weight loss $\lambda_{l^*} < \tau$. Note that there will be negative drift caused by negative weights added to initial observations as window is expanded. By controlling for weight loss, negative drift is still substantial as $\{\tilde{X}_t\}_{t=l^*+1,\ldots,T}$ are computed on an expanding window.

Method 1.4.36. *Fixed-Width Window*
Drop weights after their modulus $|\omega_k|$ decreases below a given threshold τ. This is equivalent to finding the first l^* such that $|\omega_{l^*}| \geq \tau$ and $|\omega_{l^*+1}| \leq \tau$, setting a new variable $\tilde{\omega}_k$:

$$\tilde{\omega_k} = \begin{cases} \omega_k & \text{if } k \leq l^* \\ 0 & \text{if } k > l^* \end{cases} \quad , \qquad \tilde{X}_t = \sum_{k=0}^{l^*} \tilde{\omega}_k X_{t-k} \ \text{ for } t = T - l^* + 1, \ldots, T$$

Note that the same vector of weights is used across all estimates of $\{\tilde{X}_t\}_{t=l^*,\ldots,T}$, hence avoiding negative drift caused by expanding window's added weights.
Distribution has skewness and excess kurtosis from memory, but it is stationary.

2 Mathematical and Statistical Foundation

An overview of the following fields of math:

i. Time Series Analysis

ii. Stochastic Processes

iii. Machine Learning

This section serves as a guide in providing the fundamental knowledge required.

2.1 Time Series Analysis

Based on the books by James Douglas Hamilton (1994), and ...

2.1.1 Stationary Time Series

Definition 2.1.1. A *linear first-order difference equation* is defined as

$$y_t = \phi y_{t-1} + w_t$$

where y_t is the target variable (with y_{t-1} the lag 1 variable), w_t is an input variable at time t

The difference equation may be solved by recursive substitution to arrive at

$$y_t = \phi^{t+1} y_{-1} + \phi^t w_0 + \phi^{t-1} w_1 + \phi^{t-2} w_2 + \cdots + \phi w_{t-1} + w_t$$

The *dynamic multiplier* calculates effect of w_t on y_{t+j}, and is given by

$$\frac{\partial y_{t+j}}{\partial w_t} = \phi^j$$

If $|\phi| < 1$, the system is stable. If $|\phi| > 1$, then the system is explosive.
We may generalise the process to *p-th order difference equation*, i.e.,

$$y_t = \phi_t y_{t-1} + \phi_2 t_{t-2} + \cdots + \phi_p y_{t-p} + w_t$$

We may rewrite this as a first-order difference equation in a $(p \times 1)$ vector $\boldsymbol{\xi}_t$:

$$\boldsymbol{\xi}_t = \begin{bmatrix} y_t \\ y_{t-1} \\ \vdots \\ y_{t-p+1} \end{bmatrix}$$

Define the $(p \times p)$ matrix $\boldsymbol{F}$ by

$$\boldsymbol{F} = \begin{bmatrix} \phi_1 & \phi_2 & \phi_3 & \cdots & \phi_{p-1} & \phi_p \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

Finally, define the $(p \times 1)$ vector $\boldsymbol{v}_t$ by

$$\boldsymbol{v}_t = \begin{bmatrix} w_t \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Then the system of p equations $\boldsymbol{\xi}_t = \boldsymbol{F}\boldsymbol{\xi}_{t-1} + \boldsymbol{v}_t$ is identical to the first order difference equation, and by recursive substitution, we have the following for the case of p-th order difference equation:

$$\boldsymbol{\xi}_{t+j} = \boldsymbol{F}^{j+1}\boldsymbol{\xi}_{t-1} + \boldsymbol{F}^j \boldsymbol{v}_t + \boldsymbol{F}^{j-1}\boldsymbol{v}_{t+1} + \cdots + \boldsymbol{F}\boldsymbol{v}_{t+j-1} + \boldsymbol{v}_{t+j}$$

Hence the dynamic multiplier is then $\frac{\partial y_{t+j}}{\partial w_t} f_{11}^{(j)}$, where $f_{11}^{(j)}$ is the $(1,1)$ element of $\boldsymbol{F}^j$.

2.1.2 Univariate Time Series Models

Trading activities through an exchange can be described by a sequence of time stamps ('ticks') $t_0 < t_1 < \cdots < t_n$, and 'marks' y_i at time t_i, where t_i denote market open, t_n denote market close. The marks y_i is a characteristic of the order book at time of ith activity. Events with marks associated with the ticks can be described mathematically as a marked point process.

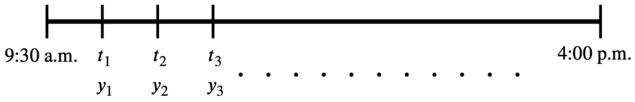

Figure 5: Ticks and Marks

A data aggregation method is where aggregation is conducted when there is a change in marker. An alternative aggregation method would be to divide the time span T for exchange hours into K intervals, so regularly spaced intervals are of size $\Delta t = T/K$. The time series method to be discussed will apply to all aggregated data.

Let $p_{it} = \ln P_{it}$ denote price of ith asset at time t; $p_t = (p_{1t}, p_{2t}, \ldots, p_{nt})$ denote price vector for n assets; y_{it} denote vector of characteristics of ith asset at time t. These quantities are aggregated from high frequency data. Consider r factors $f_t = (f_{1t}, f_{2t}, \ldots, f_{rt})$ that may include market and industry factors, and asset characteristics. Trading rules can be broadly grouped as follows:

i. *Statistical Arbitrage*: $E(p_{i,t+1} \mid p_{i,t}, p_{i,t-1}, \ldots, y_{i,t}, y_{i,t-1}, \ldots)$, that predicts the price of ith stock at $t+1$ based on past trading information (also known as time series momentum)

ii. *Momentum*: $E(p_{t+1} \mid p_t, p_{t-1}, \ldots, y_t, y_{t-1}, \ldots)$, that predicts the cross-sectional momentum of a subset of stocks based on past trading characteristics. For portfolio formation and rebalancing, pairs trading

iii. *Fair Value*: $E(p_{t+1} \mid p_t, p_{t-1}, \ldots, y_t, y_{t-1}, \ldots, f_t, f_{t-1}, \ldots)$, predicts price using all relevant quantities. Factors normally include market, Fama-French; at a more macro level than timescale considered for price prediction, but may still be useful.

Hence price and volatility prediction can be formulated as a time series prediction problem. Autocorrelations and partial autocorrelations can be used to build autoregressive and ARCH models with some predictive power.

Let $Y_1, Y_2, \ldots, Y_T$ be a sequence of random variables with a joint probability distribution. A sequence of observations of stochastic process $\{Y_t, t = 1, \ldots, T\}$ is a realisation of the process.
A time series $\{Y_t\}$ is *stationary* if for every integer m, the set of variables $Y_{t_1}, Y_{t_2}, \ldots, Y_{t_m}$ depends only on the distance between times $t_1, t_2, \ldots, t_m$. Thus $E(Y_t) = \mu$, $Var(Y_t) = \sigma^2$ are constant for all t.
The *auto-covariance* function is defined as

$$\gamma(s) = Cov(Y_t, Y_{t-s}) = E[(Y_t - \mu)(Y_{t-s} - \mu)] \quad \forall s = 0, \pm 1, \ldots$$

The *auto-correlation* of process at lag s is defined as

$$\rho(s) = Corr(Y_t, Y_{t-s}) = \frac{Cov(Y_t, Y_{t-s})}{(Var(Y_t)\ Var(Y_{t-s}))^{1/2}} = \frac{\gamma(s)}{\gamma(0)}, \quad \forall s = 0, \pm 1, \ldots$$

Some examples of stationary stochastic processes are as follows:

Example 2.1.2. *(White Noise)* Sequence of discrete independent random variables (r.v.) $\{\epsilon_t\}$ with $E[\epsilon_t] = 0$ and $E[\epsilon_t^2] = \sigma^2$. Set $Y_t = \mu + \epsilon_t$, then $E[Y_t] = \mu$, $Cov(Y_t, Y_{t-s}) = Var(Y_t) = \sigma^2$ if $s = 0$, and $Cov(Y_t, Y_{t-s}) = 0$ if $s \neq 0$. Hence the process is stationary.

Example 2.1.3. *(Moving Average)* Let $\{\epsilon_t\}$ be independent r.v., with process $\{Y_t\}$ where $Y_t = \mu + \epsilon_t + \epsilon_{t_1}$ for $t = 0, 1, 2, \ldots$, and μ is constant. Then $E[Y_t] = \mu$ $\forall t$, and

$$Cov(Y_t, Y_{t-s}) = \gamma(s) = \begin{cases} 2\sigma^2 & \text{if } s = 0 \\ \sigma^2 & \text{if } s = 1 \\ 0 & \text{if } s > 1 \end{cases}$$

Hence $\{Y_t\}$ is stationary, with $\rho(s) = \gamma(s)/\gamma(0)$ such that $\rho(0) = 1$, $\rho(1) = 1/2$, $\rho(s) = 0$ for $|s| > 1$.

Some examples of non-stationary stochastic processes are as follows:

Example 2.1.4. *(Random Walk with Drift)* Let $\{\epsilon_t\}$ be sequence of independent r.v. with $E[\epsilon_t] = 0$, $E[\epsilon_t^2] = \sigma^2$, and define process $\{Y_t\}$ by $Y_t = Y_{t-1} + \delta + \epsilon_t$ with $Y_0 = 0$. Then the process can be summarised as

$$Y_t = \delta t + \sum_{j=1}^{t} \epsilon_j$$

Note that $E[Y_t] = \delta t$ and $Var[Y_t] = t\sigma^2$, hence process $\{Y_t\}$ is not stationary.

Given T observations from stationary process $\{Y_t\}$, the *sample mean* is $\overline{Y} = \frac{1}{T}\sum\limits_{t=1}^{T} Y_t$. The *sample auto-covariance* function is defined by $\hat{\gamma}(s) = \frac{1}{T}\sum\limits_{t=1}^{T-s}(Y_t - \overline{Y})(Y_{T-s} - \overline{Y})$ for $s = 0, 1, \ldots$. The *sample auto-correlation* function (ACF) is defined as $\hat{\rho}(s) = \frac{\hat{\gamma}(s)}{\hat{\gamma}(0)} = r(s)$. The *sample variance* is $Var(\overline{Y}) = \frac{\gamma(0)}{T}\left[1 + 2\sum\limits_{u=1}^{Y-1}\left(\frac{T-u}{T}\right)\rho(u)\right]$; note that it has to account for auto-correlations.

Definition 2.1.5. A stochastic process $\{Y_t\}$ is a *linear process* if it can be represented as

$$Y_t = \mu + \sum_{j=0}^{\infty} \Psi_j \epsilon_{t-j}$$

where ϵ_t are independent with mean 0, variance σ_ϵ^2, and $\sum\limits_{j=0}^{\infty} < \infty$.

2.2 Classical Machine Learning

2.2.1 Ensemble Methods

2.3 Deep Learning

2.3.1 Deep Feedforward Networks

Remark 2.3.1. *Deep Feedforward Networks (DFNs)*
Defines a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns value of parameters $\boldsymbol{\theta}$ that results in best function approximation.

Remark 2.3.2. *Linear Models*
Linear models may fit data efficiently and reliably, either in closed form or with convex optimisation.
Model capacity limited to linear functions, does not model interaction between any two input variables

Remark 2.3.3. *Nonlinear Models*
To represent nonlinear functions of $\boldsymbol{x}$, apply linear model to transformed input $\phi(\boldsymbol{x})$ where ϕ is a nonlinear transformation. Kernel trick may be applied to obtain nonlinear learning algorithm. To choose mapping ϕ:

i. Choose generic ϕ, such as that used by kernel machines based on RBF kernel.
If $\phi(\boldsymbol{x})$ is of high enough dimension, can find enough capacity to fit training set, but generalisations to test set remains poor. Mappings are based on principle of local smoothness and do not encode enough prior information to solve advanced problems.

ii. Manually engineer ϕ. Requires decades of human effort for each task, and practitioners specialising in different domains (i.e, speech recognition, computer vision) with little transfer between domains.

iii. Learn ϕ through deep learning. Model is $y = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}; \boldsymbol{\theta})^T \boldsymbol{w}$, where parameters $\boldsymbol{\theta}$ can be used to learn ϕ from broad class of functions, and parameters $\boldsymbol{w}$ that map from $\phi(\boldsymbol{x})$ to desired output.
Do not require training problem to be convex, and only require finding the right general function.

Definition 2.3.4. *Cost Functions*

i. Learning Conditional Distributions with Maximum Likelihood: cost function is negative log-likelihood, which is the cross-entropy between training data and model distribution:

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x})$$

Specific form of cost function changes from model to model, depending on form of $\log p_{\text{model}}$.
Method removes the burden of designing cost functions for each model, as specifying a model $p(\boldsymbol{y} \mid \boldsymbol{x})$ automatically determines a cost function $\log p(\boldsymbol{y} \mid \boldsymbol{x})$.
Gradient of cost function for neural networks must be large and predictable enough to serve as good guide for the learning algorithm. The negative log-likelihood helps to avoid saturation of the function.

ii. Learning Conditional Statistics: to learn just one conditional statistic of $\boldsymbol{y}$ given $\boldsymbol{x}$.
With sufficiently powerful neural network, this can represent any function f from a wide class of function, limited by features of continuity and boundedness. Hence the cost function is a functional. Learning is choosing a functional rather than a set of parameters.
By calculus of variation , solving the optimisation problem yields the below function,

$$f^* = \arg\min_f \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x})\|^2$$
$$f^* = \mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}(\boldsymbol{y} \mid \boldsymbol{x})}[\boldsymbol{y}]$$

If infinitely many samples from the true data-generating distribution could be trained, then minimising the mean squared error cost function gives a function that predicts mean of $\boldsymbol{y}$ for each value of $\boldsymbol{x}$.
A second result from calculus of variations is:

$$f^* = \arg\min_f \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x})\|_1$$

which is a function that predicts the median value of $\boldsymbol{y}$ for each $\boldsymbol{x}$. This is the mean absolute error.
Note that mean squared error and mean absolute error often lead to poor results when used with gradient-based optimisation. Output units that saturate may produce very small gradients when combined with these cost functions. Hence the reason that cross-entropy cost function is more popular.

Definition 2.3.5. *Output Units*

i. Linear Units: base on affine transformation with no nonlinearity.
Given features $\boldsymbol{h}$, a layer of output units produces vector $\hat{\boldsymbol{y}} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$. Linear output layers are used to produce mean of conditional Gaussian distribution:

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$$

Maximising log-likelihood is equivalent to minimising mean squared error.
Linear units do not saturate, hence may be used for wide variety of optimisation algorithms.

ii. Sigmoid Units: define Bernoulli distribution y conditioned on $\boldsymbol{x}$. Neural net to predict $P(y = 1 \mid \boldsymbol{x})$, which lies in interval $[0, 1]$. The sigmoid unit is defined by

$$\hat{y} = \sigma(\boldsymbol{w}^T \boldsymbol{h} + b)$$

where σ is the logistic sigmoid function.
Note that the cost function used with maximum likelihood is $-\log P(y \mid \boldsymbol{x})$, preventing saturation.
The loss function for MLE of Bernoulli parametrised by sigmoid is

$$J(\boldsymbol{\theta}) = -\log P(y \mid \boldsymbol{x}) = -\log \sigma((2y-1)z) = \zeta((1-2y)z)$$

Function saturates only when $(1 - 2y)z$ is very negative, i.e., when model has the right answer.

iii. Softmax Units: used to represent probability distribution over n different classes.
A linear layer predicts unnormalised log probabilities:

$$\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{h} + \boldsymbol{b}$$

where $z_i = \log \tilde{P}(y = i \mid \boldsymbol{x})$. Softmax function is then

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

The function is to maximise $\log P(y = i; \boldsymbol{z}) = \log \text{softmax}(\boldsymbol{z})_i = z_i - \log \sum_j \exp(z_j)$.
Note that the input z_i always has direct contribution to cost function, and the term cannot saturate.
Un-regularised maximum likelihood will drive the model to learn parameters that drive the softmax to predict fraction of counts for each outcome observed in the training set:

$$\text{softmax}(z(\boldsymbol{x}; \boldsymbol{\theta}))_i \approx \frac{\sum_{j=1}^{m} \mathbf{1}_{y^{(j)}=i, \boldsymbol{x}^{(j)}=x}}{\sum_{j=1}^{m} \mathbf{1}_{\boldsymbol{x}^{(j)}=x}}$$

Note that objective functions other than log-likelihood does not work well with softmax function. Squared error is poor loss function for softmax units, and can fail to train the model to change its output.
Softmax function can saturate, and many functions based on softmax also saturate, unless they are able to invert the saturating activating function.

iv. Other Output Types: generally, given a conditional distribution $p(\boldsymbol{y} \mid \boldsymbol{x};\ \boldsymbol{\theta})$, principle of maximum likelihood suggests using $-\log p(\boldsymbol{y} \mid \boldsymbol{x};\ \boldsymbol{\theta})$ as the cost function.
Neural networks represent a function $f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{\omega}$. The outputs of the function are not direct predictions of value $\boldsymbol{y}$, but the parameters for distribution over $\boldsymbol{y}$. The loss function is then $-\log p(\boldsymbol{y}; \boldsymbol{\omega}(\boldsymbol{x}))$.

Remark 2.3.6. *Learning Distribution Parameters*

i. Heteroscedastic Model: to predict different variance in $\boldsymbol{y}$ for different values of $\boldsymbol{x}$, formulate the Gaussian distribution using precision rather than variance. In multi-variate case, the diagonal precision matrix is used, $\text{diag}(\boldsymbol{\beta})$. The log-likelihood of Gaussian distribution parametrised by $\boldsymbol{\beta}$ involves only multiplication by β_i and addition of $\log \boldsymbol{\beta}_i$. The gradient of these operations are well-behaved.
Let $\boldsymbol{\alpha}$ be raw activation of the model used to determine diagonal precision. The softplus function may be used to obtain a positive precision vector $\boldsymbol{\beta} = \zeta(\boldsymbol{\alpha})$. Same strategy applies equally if using variance variance or standard deviation rather than precision.
If covariance is full and conditional, then parametrisation must be chosen that guarantees positive-definiteness of predicted covariance matrix.

$$\boldsymbol{\Sigma}(\boldsymbol{x}) = \boldsymbol{B}(\boldsymbol{x}) \boldsymbol{B}^T(\boldsymbol{x})$$

where $\boldsymbol{B}$ is unconstrained square matrix. Note that if matrix is full rank, then computing likelihood requires $O(d^3)$ a $d \times d$ matrix for the determinant and inverse of $\boldsymbol{\Sigma}(\boldsymbol{x})$.

ii. Mixture Density Networks: to perform multimodal regression (predict real values that come from conditional distribution $p(\boldsymbol{y} \mid \boldsymbol{x})$) that can have several different peaks in $\boldsymbol{y}$ for the same $\boldsymbol{x}$.

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{i=1}^{n} p(c = i \mid \boldsymbol{x}) \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}^{(i)}(\boldsymbol{x}), \boldsymbol{\Sigma}^{(i)}(\boldsymbol{x}))$$

The neural network will have three outputs:

1. Mixture components $p(c = i \mid \boldsymbol{x})$, forming a multinoulli distribution over n different components with latent variable c, obtained by softmax over n-dimensional vector.
2. Means $\boldsymbol{\mu}^{(i)}(\boldsymbol{x})$
3. Covariances $\boldsymbol{\Sigma}^{(i)}(\boldsymbol{x})$

Gradient-based optimisation of conditional Gaussian mixtures can be unreliable as the divisions can be numerically unstable. May be solved by clipping gradients, or scaling gradients heuristically.

Definition 2.3.7. *Hidden Units*
Even if hidden units are not differentiable at all input points, gradient descent still performs well enough as the training algorithms do not usually arrive at local minimum of cost function, but reduce its value significantly. Most hidden units has input vector $\boldsymbol{x}$, computes an affine transformation $\boldsymbol{z} = \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}$, then applying element-wise non-linear function $g(\boldsymbol{z})$.

i. Rectified Linear Units (ReLUs): uses activation function $g(z) = \max\{0, z\}$.
Typically used on top of an affine transformation $\boldsymbol{h} = g(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$. In initialisation, set all elements of $\boldsymbol{b}$ to small positive values, so that ReLUs will be initially active for most inputs in training set.
Generalisations of ReLUs have non-zero slope α_i for $z_i < 0$: $h_i = \max(0, z_i) + \alpha_i \min(0, z_i)$.

   1. Absolute Value Rectification: sets $\alpha_i = -1$ to obtain $g(z) = |z|$. Used for object recognition from images, to seek features that are invariant under polarity reversal of input illumination.
   2. Leaky ReLU: fixes α_i to small positive value
   3. Parametric ReU: treats α_i as a learnable parameter

ii. Maxout Units: divide $\boldsymbol{z}$ into groups of k values. Each maxout unit then outputs maximum element of one of these groups: $g(\boldsymbol{z})_i = \max_{j \in \mathbb{G}^{(i)}} z_j$, where $\mathbb{G}^{(i)}$ is indices of inputs for group i, which is $\{(i-1)k+1, \ldots, ik\}$.
This allows learning of piecewise linear function that responds to multiple directions in input $\boldsymbol{x}$ space .
Maxout units learn the activation function itself. With large k, maxout unit can learn to approximate any convex function. Each maxout unit is parametrised by k weight vectors, hence need more regularisation than ReLUs. Benefits include:

   1. Can work well without regularisation if training set is large and number of pieces per unit is low.
   2. Can gain statistical and computation advantages by requiring fewer parameters.
   3. Have redundancy that resists catastrophic forgetting, where neural networks forgot how to perform tasks that were trained on in the past.

iii. Logistic Sigmoid and Hyperbolic Tangent: the logistic sigmoid activation function is $g(z) = \sigma(z)$, and the hyperbolic tangent activation function is $g(z) = \tanh(z)$. Note that $\tanh(z) = 2\sigma(2z) - 1$.
Sigmoidal units saturate across most of the domain, which makes gradient-based learning very difficult. Hence the use in hidden units in feedforward networks is now discouraged.
If sigmoidal activation function must be used, hyperbolic tangent activation function performs better, as it resembles identity function more closely. Training $\hat{y} = \boldsymbol{w}^T \tanh(\boldsymbol{U}^T \tanh(\boldsymbol{V}^T\boldsymbol{x}))$ resembles training a linear model $\hat{y} = \boldsymbol{w}^T\boldsymbol{U}^T\boldsymbol{V}^T\boldsymbol{x}$ as long as the activations of the network can be kept small.
Sigmoidal functions are more common in recurrent networks, probabilities models, auto-encoders

iv. Linear Unit: consider neural network layer with n inputs, p outputs, $\boldsymbol{h} = g(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$. Replace with two layers, one using weight matrix $\boldsymbol{U}$ and the other using weight matrix $\boldsymbol{V}$. If first layer has no activation function, then the factored approach is to compute $\boldsymbol{h} = g(\boldsymbol{V}^T\boldsymbol{U}^T\boldsymbol{x} + \boldsymbol{b})$. If $\boldsymbol{U}$ produces q outputs, then $\boldsymbol{U}$ and $\boldsymbol{V}$ together only contains $(n + p)q$ parameters, while $\boldsymbol{W}$ contains np parameters. For small q, this is considerable saving in parameters, while the cost is constraining linear transformation to be low rank. This is an efficient way of reducing number of parameters in the model.

v. Softmax: naturally represent probability distribution over discrete variable with k possible values. Used only in more advanced architectures that explicitly learn to manipulate memory.

vi. Radial Basis Function (RBF): function becomes more active as $\boldsymbol{x}$ approaches a template $\boldsymbol{W}_{:,i}$. As it saturates to 0 for most $\boldsymbol{x}$, it can be difficult to optimise.

$$h_t = \exp\left(-\frac{1}{\sigma_i^2}\|\boldsymbol{W}_{:,i} - \boldsymbol{x}\|^2\right)$$

vii. Softplus: smooth version of rectifier for functional approximation and for conditional distributions of undirected probabilistic models. Usage is generally discouraged.

$$g(a) = \zeta(a) = \log(1 + e^a)$$

viii. Hard tanh: shaped similarly to tanh but bounded.

$$g(a) = \max(-1, \min(1, a))$$

Theorem 2.3.8. *Universal Approximation Theorem*
A feedforward network with linear output layer and at least one hidden layer with any 'squashing' activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided the network is given enough hidden units.

Method 2.3.9. *Architecture Design*
Neural network layers are arranged in a chain structure, with each layer being a function of preceding layer.

$$\begin{aligned}\boldsymbol{h}^{(1)} &= g^{(1)}(\boldsymbol{W}^{(1)T}\boldsymbol{x} + \boldsymbol{b}^{(1)})\\ \boldsymbol{h}^{(k)} &= g^{(k)}(\boldsymbol{W}^{(k)T}\boldsymbol{h}^{(k-1)} + \boldsymbol{b}^{(k)}), \quad k \geq 2\end{aligned}$$

The main considerations are the depth of network and width of each layer. Deeper networks use far fewer units per layers and far fewer parameters, and often generalise to the test set, but are harder to optimise.

Remark 2.3.10. *Depth of Network and Universal Approximation Theorem*
A feedforward network with single layer is sufficient to represent any function, but the layer may be infeasibly large and fail to learn and generalise correctly. Using deeper models can reduce number of units required to represent the desired function and can reduce generalisation error.
Shallow networks with broad family of non-polynomial activation functions have universal approximation properties. Piecewise linear networks can represent functions with number of regions that is exponential to depth.
The number of linear regions carved out by deep rectifier network with d inputs, depth l, and n units per hidden layer is $O(\binom{n}{d}^{d(d-1)}n^d)$. For maxout networks with k filters per unit, this is $O(k^{(k-1)+d})$.

Remark 2.3.11. *Backpropagation*
To calculate the gradient of loss function with respect to each of individual parameters of the neural network. Model training begins with random initialisation of weights and biases.

i. Forward pass: input is sampled from training data. Nodes receive input vector and passes their value (multiplied by random initial weight) to nodes of first hidden layer. The hidden units take weighted sum of these output values as an input to the activation function, whose output is used for next hidden layer.

ii. Error computation: the final output of the network is compared to the ground truth, difference is calculated for the error value.

iii. Backwards pass: the error value computed earlier is used to compute the gradient of loss function. The gradient is then propagated back through the network, and the weights are updated according to their contribution to the error. The learning rate determines the size of weight updates.

iv. Weights update: the weights are updated in opposite direction of the gradient

Algorithm 1: Backpropagation Learning Algorithm

```
Require:
  A set of training examples $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$
  A multilayer network with $L$ layers, weights $w_{ij}^l$, and activation function $f$
  Loss function $J(y, o)$
  Learning rate $0 < \alpha < 1$
  Number of epochs epochs.

  Initialize all weights:
  for $w_{ij}^l$ in the network do
      $w_{ij}^l \leftarrow$ small random number
  end for

  for $i = 1$ to epochs do
      for all training examples $(x, y) \in \mathcal{D}$ do
```

```
    *Propagate the inputs forward to compute the outputs*
    for all neurons $i$ in the input layer do
        $a_i^0 \leftarrow x_i$
    end for

    for $l = 2$ to $L$ do
        for all neurons $i$ in layer $l$ do
            $z_i^l \leftarrow \sum_j w_{ij}^l \, a_j^{l-1}$
            $a_i^l \leftarrow f\left(z_i^l\right)$
        end for
    end for

    *Propagate deltas backward from output layer to input layer*
    for all neurone $i$ in the output layer do
        *Let $o_i \equiv a_i^L$ be output of neurone $i$*
        $\delta_i^L \leftarrow \frac{\partial J(y_i, o_i)}{\partial o_i} f'\left(z_i^L\right)$
    end for

    for $l = L-1$ to $1$ do
        for all neurons $i$ in layer $l$ do
            $\delta_i^l \leftarrow f'\left(z_i^l\right) \sum_j \left(w_{ji}^{l+1} \, \delta_j^{l+1}\right)$
        end for
    end for

    *Update weights using the deltas*
    for all $w_{ij}^l$ in the network do
        $w_{ij}^l \leftarrow w_{ij}^l - \alpha \, \delta_i^l \, a_j^{l-1}$
    end for
  end for
end for
```

Example 2.3.12. *Backpropagation Computation Example (Matt Mazur (2015))*

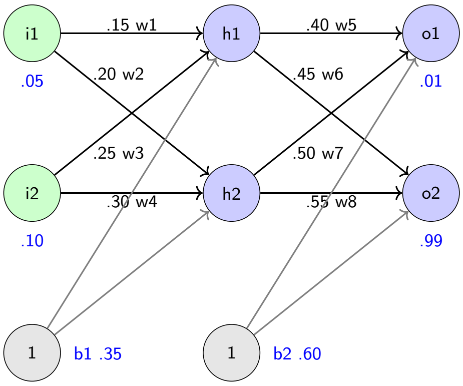

Figure 6: Backpropagation Example

For the forward pass, compute the total net input to each hidden layer neurone, squash total net input with activation function (logistic function), then repeat process with output layer neurones.
The total net input for h_1 is:

$$\begin{aligned} \text{net}_{h1} &= w_1 \cdot i_1 + w_2 \cdot i_2 + b_1 \cdot 1 \\ &= 0.15 \cdot 0.05 + 0.2 \cdot 0.1 + 0.35 \cdot 1 = 0.3775 \end{aligned}$$

Squash with logistic function to get output of h_2:

$$\text{out}_{h1} = 1/(1+e^{-\text{net}_{h1}}) = 1/(1+e^{-0.3775}) = 0.593269992$$

For h_2, repeating the same process, the output is then $\text{out}_{h2} = 0.596884378$.
Repeating the process for outer layer neurones,

$$\begin{aligned}\text{net}_{o1} &= w_5 \cdot \text{out}_{h1} + w_6 \cdot \text{out}_{h2} + b_2 \cdot 1 \\ &= 0.4 \cdot 0.593269992 + 0.45 \cdot 0.596884378 + 0.6 \cdot 1 = 1.105905967 \\ \text{out}_{o1} &= 1/(1+e^{-\text{net}_{o1}}) = 1/(1+e^{-1.105905967}) = 0.75136507\end{aligned}$$

For o_2, repeating the same process, the output is then $\text{out}_{o2} = 0.772928465$.
The total error for each output neurone may be computed using squared error function $E_{\text{total}} = \sum \frac{1}{2}(Y - \hat{Y})^2$. For o_1, the target output is 0.01, neurone output is 0.75136507, hence error is

$$E_{o1} = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

Repeating the process for o_2, the total error for neural network is then

$$E_{\text{total}} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

For the backpropagation step, update each of the weights in network to reduce the error for each output neurone.

$$\begin{aligned}E_{\text{total}} &= \frac{1}{2}(Y_{o1} - \hat{Y}_{o1})^2 + \frac{1}{2}(Y_{o2} - \hat{Y}_{o2})^2 \\ \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} &= 2 \cdot \frac{1}{2}(Y_{o1} - \hat{Y}_{o1}) \cdot -1 + 0 = -(0.01 - 0.75136507) = 0.74136507\end{aligned}$$

Also, note that

$$\begin{aligned}\text{out}_{o1} &= 1/(1+e^{-\text{net}_{o1}}) \\ \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} &= \text{out}_{o1}(1 - \text{out}_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602 \\ \text{net}_{o1} &= w_5 \cdot \text{out}_{h1} + w_6 \cdot \text{out}_{h2} + b_2 \cdot 1 \\ \frac{\text{net}_{o1}}{w_5} &= 1 \cdot \text{out}_{h1} \cdot w_5^0 + 0 + 0 = \text{out}_{h1} = 0.593269992\end{aligned}$$

Putting it all together,

$$\begin{aligned}\frac{\partial E_{\text{total}}}{\partial w_5} &= \frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} \cdot \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} \cdot \frac{\partial \text{out}_{o1}}{\partial w_5} \\ &= -(Y_{o1} - \hat{Y}_{o1}) \cdot \text{out}_{o1}(1 - \text{out}_{o1}) \cdot textout_{o1} \\ &= 0.74136507 \cdot 0.186815602 \cdot 0.593269992 = 0.0821267041\end{aligned}$$

To decrease the error, subtract the value from current weight multiplied by learning rate (i.e., $\alpha = 0.5$).

$$w_5^+ = w_5 - \alpha \frac{\partial E_{\text{total}}}{\partial w_5} = 0.4 - 0.5 \cdot 0.0821267041$$

Repeating for the rest of the weights, we get

$$w_6^+ = 0.408666186, \quad w_7^+ = 0.511301270, \quad w_8^+ = 0.561370121$$

For the hidden layers, note that

$$\begin{aligned}\frac{\partial E_{\text{total}}}{\partial w_1} &= \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \cdot \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \cdot \frac{\partial \text{net}_{h1}}{\partial w_1} \\ &= \left(\frac{\partial E_{o1}}{\partial \text{out}_{h1}} + \frac{\partial E_{o2}}{\partial \text{out}_{h1}}\right) \cdot \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \cdot \frac{\partial \text{net}_{h1}}{\partial w_1}\end{aligned}$$

Note that out_{h1} affects both out_{o1} and out_{o2}.

$$\begin{aligned}
\frac{\partial E_{o1}}{\partial \text{out}_{h1}} &= \frac{\partial E_{o1}}{\partial \text{net}_{h1}} \cdot \frac{\partial \text{net}_{h1}}{\partial \text{out}_{h1}} \\
\frac{\partial E_{o1}}{\partial \text{net}_{h1}} &= \frac{\partial E_{o1}}{\partial \text{out}_{h1}} \cdot \frac{\partial \text{out}_{o1}}{\partial \text{net}_{h1}} = 0.74136507 \cdot 0.186815602 = 0.138498562 \\
\frac{\partial \text{net}_{h1}}{\partial \text{out}_{h1}} &= w_5 = 0.4 \\
\frac{\partial E_{o1}}{\partial \text{out}_{h1}} &= 0.138498562 \cdot 0.4 = 0.055399425
\end{aligned}$$

Following same process, $\frac{\partial E_{o2}}{\partial \text{out}_{h1}} = -0.019049119$. Hence

$$\frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} = \frac{\partial E_{o1}}{\partial \text{out}_{h1}} + \frac{\partial E_{o2}}{\partial \text{out}_{h1}} = 0.05539425 - 0.019049119 = 0.036350306$$

Next, figure out $\frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}}$ and $\frac{\partial \text{net}_{h1}}{\partial w_1}$.

$$\begin{aligned}
\frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} &= \text{out}_{h1} \cdot (1 - \text{out}_{h1}) = 0.59326999(1 - 0.59326999) = 0.2410.241300709 \\
\frac{\partial \text{net}_{h1}}{\partial w_1} &= i_1 = 0.05
\end{aligned}$$

Putting it together,

$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \cdot \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \cdot \frac{\partial \text{net}_{h1}}{\partial w_1} = 0.036350306 \cdot 0.241300709 \cdot 0.05 = 0.000438658$$

This is also written as

$$\frac{\partial E_{\text{total}}}{\partial w_1} = \left(\sum_o \frac{\partial E_{\text{total}}}{\partial \text{out}_o} \cdot \frac{\partial \text{out}_o}{\partial \text{net}_o} \cdot \frac{\partial \text{net}_o}{\partial \text{out}_{h_1}} \right) \cdot \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \cdot \frac{\partial \text{net}_{h1}}{\partial w_1}$$

Next, update w_1 and rest of the weights

$$\begin{aligned}
w_1^+ &= w_1 - \alpha \frac{\partial E_{\text{total}}}{\partial w_1} = 0.15 - 0.5 \cdot 0.000438658 = 0.149780716 \\
w_2^+ &= 0.19956143, \quad w_3^+ = 0.24975114, \quad w_4^+ = 0.29950229
\end{aligned}$$

Hence all the weights has been updated. After the first round of backpropagation, total error has decreased to 0.291027924. Repeating the process 10,000 times, the error reduces to 0.0000351085.

2.3.2 Regularisation for Deep Learning

Definition 2.3.13. *Regularisation* refers to adding a parameter norm penalty $\Omega(\boldsymbol{\theta})$ to the objective function J. The regularised objective function is then

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

where $\alpha in[0, \infty)$ is a hyper-parameter that weights the contribution of norm penalty term.
For neural networks, the parameter norm penalty Ω is chosen such that it penalises only the weights of the affine transformation at each layer and leaves the biases un-regularised.

Definition 2.3.14. *L^2 (Ridge) Regularisation*
The regularisation term added to objective function is

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

Remark 2.3.15. *Behaviour of Weight Decay (L^2) Regularisation*
Assuming no bias parameter, a model have the following objective function and parameter gradient:

$$\begin{aligned} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) &= \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \\ \nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) &= \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \end{aligned}$$

On a single gradient step, the update is as follows:

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

The addition of weight decay has modified learning rule to multiplicatively shrink weight vector by constant factor on each step before gradient update.
Using quadratic approximation to objective function at minimal unregularised training cost, approximation is

$$\hat{J}(\overline{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)\boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*), \qquad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$

where $\boldsymbol{H}$ is Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$. Minimum of $\hat{J}$ occurs where gradient is

$$\nabla_{\boldsymbol{w}}\hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) = \boldsymbol{0}$$

Adding weight decay gradient, solve for minimum of regularised version of $\hat{J}$. Let $\tilde{\boldsymbol{w}}$ be minimum, then

$$\begin{aligned} \alpha\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) &= 0 \\ (\boldsymbol{H} + \alpha\boldsymbol{I})\tilde{\boldsymbol{w}} &= \boldsymbol{H}\boldsymbol{w}^* \\ \tilde{\boldsymbol{w}} &= (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^* \end{aligned}$$

As $\alpha \to 0$, regularised solution $\tilde{\boldsymbol{w}} \to \boldsymbol{w}^*$. Note $\boldsymbol{H}$ is real and symmetric, hence decompose into diagonal matrix $\boldsymbol{A}$ and orthonormal basis of eigenvectors $\boldsymbol{Q}$ such that $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$, to get

$$\tilde{\boldsymbol{w}} = (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^* = \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^*$$

The effect of weight decay rescale $\boldsymbol{w}^*$ along axes defined by eigenvectors of $\boldsymbol{H}$. The component of $\boldsymbol{w}^*$ aligned with i-th eigenvector of $\boldsymbol{H}$ is rescaled by factor $\frac{\lambda_i}{\lambda_i+\alpha}$. For components where $\lambda_i \gg \alpha$, the effects of regularisation is relatively small. For components where $\lambda_i \ll \alpha$, components will be shrunk to nearly zero magnitude.

Definition 2.3.16. *L^1 Regularisation*
The L^1 regularisation is the sum of absolute values of individual parameters.

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

Definition 2.3.17. *Behaviour of L^1 Regularisation*
Assuming no bias parameter, a model has the following objective function and parameter gradient:

$$\begin{aligned} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) &= \alpha\|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \\ \nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) &= \alpha\ \mathrm{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \end{aligned}$$

Note that the regularisation contribution to gradient is a constant factor with sign equal to $\text{sign}(w_i)$. Minimum of $\tilde{J}$ occurs at where

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

Assuming the Hessian is diagonal, $\boldsymbol{H} = \text{diag}([H_{1,1}, \dots, H_{n,n}])$, where each $H_{i,i} > 0$. The quadratic approximation of regularised objective function is then

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i} (\boldsymbol{w}_i - \boldsymbol{w}_i^*)^2 + \alpha \left| w_i \right| \right]$$

$$w_i = \text{sign}(w_i^*) \max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

For the case where $w_i^* > 0$ for all i, then

i. if $w_i^* \leq \frac{\alpha}{H_{i,i}}$, the optimal value is $w_i = 0$ as contribution of $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ to regularised objective $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is overwhelmed in direction i by L^1 regularisation which pushes w_i to zero.

ii. if $w_i^* > \frac{\alpha}{H_{i,i}}$, regularisation shifts optimal value of w_i in the direction by $\frac{\alpha}{H_{i,i}}$.

For the case where $w_i^* < 0$, this happens similarly, but with L^1 penalty decreasing w_i by $\frac{\alpha}{H_{i,i}}$, with min value 0. Note that L^1 produces a more sparse solution, which is a feature selection mechanism.

Remark 2.3.18. *Norm Penalties as Constrained Optimisation*
Let cost function regularised by parameter norm penalty be

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha \Omega(\boldsymbol{\theta})$$

Construct a generalised generalised Lagrange function,

$$\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha(\Omega(\boldsymbol{\theta}) - k)$$

The solution to the constrained problem is then

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \alpha)$$

Note that optimal value α^* will shrink $\Omega(\boldsymbol{\theta})$, but not such that it is less than k.
Fixing α^*, the problem is then a function of $\boldsymbol{\theta}$,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \alpha^*) = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha^* \Omega(\boldsymbol{\theta})$$

This is the regularised training problem of minimising $\tilde{J}$. The parameter norm penalty is imposing a constraint on the weights. If explicit constraints are to be used rather than penalties, use stochastic gradient descent on $J(\boldsymbol{\theta})$, then take the projected $\boldsymbol{\theta}$ to the nearest point that satisfies $\Omega(\boldsymbol{\theta}) < k$. This method is used if k is predefined by user, and time is not to be spent on searching for α value that corresponds to this k.
Explicit constraints and re-projection work better in circumstances where non-convex optimisation is involved, and this may avoid getting stuck in local minima. Explicit constraints also impose stability, and together with high learning rate allows rapid exploration of parameter space.

Remark 2.3.19. *Dataset Augmentation*
Effective technique for object classification. Operations such as translation of images by few pixels in each direction can greatly improved generalisation, even if the model has been designed to be partially translation invariant using convolution and pooling techniques. Rotation and scaling of images are also quite effective.
For speech recognition tasks, noice injection will improve performance. To improve robustness, train models with random noise applied to inputs. Noise injection also works when applied to the hidden units (data augmentation at multiple levels of abstraction). Enabling dropout constructs new inputs by multiplying by noise.

Remark 2.3.20. *Noise Robustness*
For some models, addition of noise with infinitesimal variance at input of model is equivalent to imposing penalty on norm of the weights. Generally, noise injection can be much more powerful than simply shrinking the parameters, especially when noise is added to hidden units.
Noise may also be added to weights, which is primarily used in recurrent neural networks. This is a stochastic implementation of Bayesian inference over the weights (model weights are uncertain and representable via a probability distribution that reflects this uncertainty).

Adding noise to weights may also result in more stability in function to be learned. Let the cost function be

$$J = \mathbb{E}_{p(x,y)}[(\hat{y}(\boldsymbol{x}) - y)^2]$$

where $\hat{y}(\boldsymbol{x})$ is function to be trained that maps set of features $\boldsymbol{x}$ to a scalar using least-squares cost function J. Assume each input presentation include a random perturbation $\epsilon_{\boldsymbol{W}} \sim \mathcal{N}(\boldsymbol{\epsilon}; \boldsymbol{0}, \eta \boldsymbol{I})$ in the network weight of a standard l-layer MLP. Let the perturbed model be $\hat{y}_{\boldsymbol{\epsilon_W}}(\boldsymbol{x})$. With injection o noise, objective function becomes

$$\tilde{J}_{\boldsymbol{W}} = \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon_W})}\left[(\hat{y}_{\boldsymbol{\epsilon_W}}(\boldsymbol{x}) - y)^2\right] = \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon_W})}\left[\hat{y}^2_{\boldsymbol{\epsilon_W}}(\boldsymbol{x}) - 2y\hat{y}_{\boldsymbol{\epsilon_W}} + y^2\right]$$

For small η, the minimisation of J with added noise (with covariance $\eta \boldsymbol{I}$) is equivalent to minimisation of J with additional regularisation term $\eta \mathbb{E}_{p(\boldsymbol{x}),y}[\|\nabla_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\|^2]$. This encourages parameters to go to regions of parameter space where small perturbations of weights have relatively small influence on output.

Remark 2.3.21. *Label Smoothing*
Datasets may have some mistake in y labels. Regularise a model based on softmax with k output values by replacing hard 0 and 1 classification targets with targets of $\frac{\epsilon}{k-1}$ and $1-\epsilon$ respectively, assuming that the training set label y is correct with probability $1 - \epsilon$. Cross-entropy can then be used on these soft targets.

Remark 2.3.22. *Semi-Supervised Learning*
Unlabelled examples $P(\boldsymbol{x})$ and $P(\boldsymbol{x}, \boldsymbol{y})$ are used to estimate $P(\boldsymbol{y} \mid \boldsymbol{x})$ or predict $\boldsymbol{y}$ from $\boldsymbol{x}$.
Construct a generative model of either $P(\boldsymbol{x})$ or $P(\boldsymbol{x}, \boldsymbol{y})$ which shares parameters with a discriminative model of $P(\boldsymbol{y} \mid \boldsymbol{x})$. The supervised criterion $\log P(\boldsymbol{y} \mid \boldsymbol{x})$ may then be traded with the unsupervised or generative one ($\log P(\boldsymbol{x})$ or $\log P(\boldsymbol{x}, \boldsymbol{y})$). The generative criterion then expresses a particular form of prior belief about the solution to the supervised learning problem (structure of $P(\boldsymbol{x})$ is connected to structure of $P(\boldsymbol{y} \mid \boldsymbol{x})$ such that is is captured by the shared parametrisation).

Remark 2.3.23. *Multi-Task Learning*
Improve generalisations by pooling examples (soft constraints imposed on parameters) from several tasks.
Different supervised tasks (predicting $\boldsymbol{y}^{(i)}$ given $\boldsymbol{x}$) share the same input $\boldsymbol{x}$, as well as some intermediate-level representation $\boldsymbol{h}^{(\text{shared})}$ capturing a common pool of factors. Model can be divided in two parts:

i. Task-specific parameters
ii. Generic parameters, shared across all the tasks

Improved generalisation and associated error bounds can be achieved due to shared parameters, but this only happens if some assumptions about statistical relationship between different tasks are valid.

Remark 2.3.24. *Early Stopping*
To obtain a model with better validation set error by returning to the parameter setting at the point in time with lowest validation set error. Model validation is done in parallel to model training.
Early stopping has the effect of restricting the optimisation procedure to a relatively small volume of parameter space in neighbourhood of initial parameter value $\boldsymbol{\theta}_o$. Assume τ optimisation steps are taken with learning rate ϵ. The product $\epsilon\tau$ is a measure of effective capacity. Assuming gradient is bounded, restricting $\epsilon\tau$ limits the volume of parameter space reachable from $\boldsymbol{\theta}_o$, hence $\epsilon\tau$ behaves as the reciprocal of weight decay coefficients.

Algorithm 2: Early Stopping Algorithm

```
Require:
  Number of steps $n$ between evaluations
  Patience $p$, the number of times to observe worsening validation error before stopping
  Initial parameters $\boldsymbol{\theta}_o$

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o, \quad i \leftarrow 0, \quad j \leftarrow 0, \quad v \leftarrow \infty, \quad \boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}, \quad i^* \leftarrow i$
  while $j < p$ do
      Update $\boldsymbol{\theta}$ by running training algorithm for $n$ steps
      $i \leftarrow i + n, \quad v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
      if v' < v then
          $j \leftarrow 0, \quad \boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}, \quad i^* \leftarrow i, \quad v \leftarrow v'$
      end if
      $j \leftarrow j + 1$
  end while
  Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is $i^*$
```

Remark 2.3.25. *Parameter Tying and Parameter Sharing*
Used to express prior knowledge on suitable values of model parameters (precise values are not known, but there should be some dependencies between parameters). To regularise parameters to be close, may use:

i. Parameter norm penalty: assume model A with parameters $\boldsymbol{w}^{(A)}$ and model B with parameters $\boldsymbol{w}^{(B)}$. Two models map input to two different but related outputs $\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$ and $\hat{y}^{(B)} = g(\boldsymbol{w}^{(B)}, \boldsymbol{x})$. The model parameters $w_i^{(A)}$ should be close to $w_i^{(B)}$ $\forall i$. Using regularisation, a parameter norm penalty may be imposed, such as using L^2 penalty to get $\Omega(\boldsymbol{w}^{(A)}, \boldsymbol{w}^{(B)}) = \left\|\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}\right\|_2^2$.

ii. Parameter sharing: using constraints. Only a subset of the parameters (input set) need to be stored in memory. May lead to significant reduction in memory footprint of the model (i.e., in CCNs)

Remark 2.3.26. *Sparse Representations*
Representation sparsity is where elements of the representation are zero, accomplished by same sort of mechanisms used in parameter regularisation.
Given a linear regression $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{h}$, where $\boldsymbol{y} \in \mathbb{R}^m, \boldsymbol{B} \in \mathbb{R}^{m \times n}, \boldsymbol{h} \in \mathbb{R}^n$, and $\boldsymbol{h}$ is the representation of data $\boldsymbol{x}$, add to loss function J a norm penalty on the representation ($\Omega(\boldsymbol{h})$)):

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h})$$

where $\alpha \in [0, \infty)$ weights relative contribution of norm penalty, with larger values of α for more regularisation. L^1 penalty on elements of representation induces sparsity, $\Omega(\boldsymbol{h}) = \|\boldsymbol{h}\|_1 = \sum_i |h_i|$. For representations with elements constrained to lie on unit interval, may use Student-t prior or KL divergence penalties.
Other approaches such as hard constraint on activation values may also work. *Orthogonal matching pursuit* encodes an input $\boldsymbol{x}$ with representation $\boldsymbol{h}$ that solves the constrained optimisation problem

$$\arg \min_{\boldsymbol{h}, \|\boldsymbol{h}\|_0 < k} \|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|^2$$

where $\|\boldsymbol{h}\|_0$ is the number of non-zero entries of $\boldsymbol{h}$. If $\boldsymbol{W}$ is constrained to be orthogonal, this can be solved efficiently with OMP-k method, where k is number of non-zero features allowed.

Remark 2.3.27. *Bootstrap Aggregating (Bagging)*
Construct k different datasets, each with same number of examples as original dataset, but constructed by sampling with replacement from original dataset. Each dataset may miss some examples but also contain duplicate examples. Each model i is then trained on the dataset i. Differences in random initialisation, random selection of mini-batches, differences in hyperparameters, or different outcomes of non-deterministic implementation of neural networks are enough to allow each model to make partially independent errors.

Remark 2.3.28. *Boosting*
Construct k different datasets from original dataset. Using ith dataset, train ith weak learner. Test the ith weak learner with validation dataset, and each datapoint with wrong prediction is sent to $(i+1)$th dataset. The $(i+1)$th dataset is then used to train the $(i+1)$th weak learner. Continue until reaching the total number of subsets for the total prediction.

Remark 2.3.29. *Dropout*
Dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from the base network by multiplying its output value by zero. To train a neural network with dropout, use minibatch-based learning algorithms (i.e., Stochastic Gradient Descent). Each time an example is loaded into minibatch, randomly (and independently) sample a different binary mask $\boldsymbol{\mu}$ to apply to all of input and hidden units in the network, where the probability of sampling a mask value of one is the dropout rate.
Let $J(\boldsymbol{\theta}, \boldsymbol{\mu})$ be cost of model defined by parameters $\boldsymbol{\theta}$ and mask $\boldsymbol{\mu}$. Dropout training is minimising $\mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$. Each model inherits different subset of parameters from parent neural network, and a tiny fraction of possible sub-networks are trained for a single step. Parameter sharing allows sub-networks to arrive at good parameter settings. Each sub-network given by $\boldsymbol{\mu}$ defines a probability distribution $p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$. The arithmetic mean over all masks is then given by

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$$

where $p(\boldsymbol{\mu})$ is the distribution used to sample $\boldsymbol{\mu}$ during training.
Using dropout as a regularisation technique reduces effective capacity of a model. To offset this, increase the size of the model. Typically optimal validation error is much lower when using dropout, but this comes at cost of larger model and more iterations of training.
For very large datasets, costs of using dropout and larger models may outweigh benefits of regularisation.

For sparse datasets (extremely few labelled training examples available), Bayesian neural networks outperform dropout. When additional unlabelled data is available, unsupervised learning may gain advantage over dropout.

Remark 2.3.30. *Adversarial Training*
Adversarial example may be constructed by optimisation to search for an input $\boldsymbol{x}'$ near $\boldsymbol{x}$ such that the model output is very different at $\boldsymbol{x}'$.
The primary cause is excessive linearity in some neural networks. Changing input by ϵ causes linear function with weights $\boldsymbol{w}$ to change by as much as $\epsilon \left\|\boldsymbol{w}\right\|_1$, which can be very large if $\boldsymbol{w}$ is high-dimensional.
Adversarial examples assists in semi-supervised learning. At point $\boldsymbol{x}$ that is not associated with a label, model assigns label $\hat{y}$ (may not be true label). Seek adversarial example $\boldsymbol{x}'$ that causes classifier to output label $y' \neq \hat{y}$. Classifier may then be trained to assign same label to $\boldsymbol{x}$ and $\boldsymbol{x}'$, learning function robust to small changes.

Remark 2.3.31. *Tangent Distance Algorithm*
Non-parametric nearest-neighbour algorithm, where examples on the same manifold share the same category. The nearest-neighbour distance between points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is the distance between manifolds M_1 and M_2. Computationally cheap alternative is to approximate M_i by its tangent plane at $\boldsymbol{x}_i$ and measure the distance between two tangents, or between a tangent plane and a point.

Remark 2.3.32. *Tangent Propagation Algorithm*
Trains neural net classifier with extra penalty to make each output $f(\boldsymbol{x})$ of neural net locally invariant to factors of variation, which correspond to movement along manifold which examples from same class concentrate. Local invariance is achieved by requiring $\nabla_{\mathbf{x}} f(\boldsymbol{x})$ to be orthogonal to known manifold tangent vectors $\boldsymbol{v}^{(i)}$ at $\boldsymbol{x}$, or equivalently the directional derivative of f at $\boldsymbol{x}$ in directions $\boldsymbol{v}^{(i)}$ be small by adding regularisation penalty:

$$\Omega(f) = \sum_i \left(\left(\nabla_{\boldsymbol{x}} f(\boldsymbol{x})\right)^T \boldsymbol{v}^{(i)} \right)^2$$

Tangent vectors are derived a priori from effect of transformations (i.e., translation, rotation, scaling).
Note this regularises model to resist infinitesimal perturbation, and the infinitesimal approach poses difficulties for models based on rectified linear units.
Data augmentation is the non-infinitesimal version of Tangent Propagation Algorithm, where network is explicitly trained to correctly classify distinct inputs created from more than infinitesimal transformation.

Remark 2.3.33. *Double Back-propagation Algorithm*
Algorithm regularises Jacobian to be small. Require model to be invariant to all directions of change in input so long as the change is small. Adversarial training is the non-infinitesimal version of Double Back-propagation Algorithm, where inputs were found near original inputs and model is trained to produce same input.

Remark 2.3.34. *Manifold Tangent Classifier*
Eliminates need to know tangent vectors a priori. Auto-encoders may estimate manifold tangent vectors, and the classifier makes use of this technique to avoid requiring user-specified tangent vectors. The steps are:

i. Use auto-encoder to learn manifold structure by unsupervised learning
ii. Use these tangents to regularise a neural net classifier as in Tangent Propagation Algorithm.

2.3.3 Optimisation for Deep Learning

Remark 2.3.35. *Optimisation*
A cost function $J(\boldsymbol{\theta})$ is reduced to improve some performance measure P.
Optimisation algorithms include some specialisation on specific structure of machine learning objective functions.
Cost function is average over training set,

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

where L is per-example loss function, $f(\boldsymbol{x};\boldsymbol{\theta})$ is predicted output given input $\boldsymbol{x}$, $\hat{p}_{\text{data}}$ is empirical distribution.
For supervised learning, y is target output.
The goal is to usually to minimise objective function where expectation is taken on data generating distribution p_{data} rather than over the finite training set:

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim p_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

Note that machine learning algorithm usually minimises a surrogate loss function, and halts when convergence criterion based on early stopping is satisfied.

Definition 2.3.36. *Empirical Risk Minimisation*
To minimise expected loss on training set, where $\hat{p}(\boldsymbol{x}, y)$ is the empirical distribution. The empirical risk is

$$\mathbb{E}_{\boldsymbol{x},y\sim\hat{p}_{\text{data}}(\boldsymbol{x},y)}[L(f(\boldsymbol{x},\boldsymbol{\theta}), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$$

where m is number of training samples. Note that empirical risk minimisation is prone to overfitting, as models with high capacity may memorise the training set.

Remark 2.3.37. *Batch and Minibatch Algorithms*
Objective algorithms for machine learning computes each update to the parameters based on expected value of cost function using only a subset of terms of full cost function.
For maximum likelihood estimation problems, note that

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$
$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}, y; \boldsymbol{\theta})$$

The gradient is then

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y\sim\hat{p}_{\text{data}}} \nabla_{\boldsymbol{\theta}} \log\log p_{\text{model}}(\boldsymbol{x}, y; \boldsymbol{\theta})$$

If the optimisation algorithm uses the entire training set, then it is a batch gradient method.
If a single sample is used at a time, then this is a stochastic/online gradient method.
For algorithms that uses more than one but less than all samples, then this is a minibatch gradient method.

Remark 2.3.38. *Challenges: Ill-Conditioning of Hessian Matrix H*
May cause stochastic gradient descent to be stuck where very small step increases the cost function.
Note that a gradient descent step of $-\epsilon\boldsymbol{g}$ will add

$$\frac{1}{2}\epsilon^2\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g} - \epsilon\boldsymbol{g}^T\boldsymbol{g}$$

to the cost. This will be a problem when $\frac{1}{2}\epsilon^2\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g} > \epsilon\boldsymbol{g}^T\boldsymbol{g}$.
Monitor squared gradient norm $\boldsymbol{g}^T\boldsymbol{g}$ and $\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g}$ term. When ill-conditioning happens, gradient norm does not shrink significantly but $\boldsymbol{g}^T\boldsymbol{H}\boldsymbol{g}$ term grows more than order of magnitude, and learning rate shrinks.

Remark 2.3.39. *Challenges: Local Minimums*
Neural networks and models with multiple parametrised latent variables have multiple local minima due to *model identifiability* problem. Model is identifiable if a sufficiently large training set can rule out all but one setting of model parameters. Models with latent variables are not identifiable as equivalent models are obtained by exchanging latent variables with each other (*weight space symmetry*). If there are m layers with n units each, then there are $n!^m$ ways to arrange the hidden units, making the model unidentifiable.
There may be an extremely large or uncountably infinite amount of local minima in a cost function. If the local minima have a high cost compared to global minimum, then this is problematic.

Remark 2.3.40. *Challenges: Plateaus, Saddle Points, Flat Regions*
For a function $f : \mathbb{R}^n \to \mathbb{R}$, expected ratio of saddle points to local minima grows exponentially with n. Hessian

matrix at a saddle point has a mixture of positive and negative eigenvalues.
Degenerate regions with flat gradient may correspond to high value of objective function.

Remark 2.3.41. *Challenges: Cliffs and Exploding Gradients*
On extremely steep cliff structure, gradient update step may move parameters extremely far, losing most of the optimisation work done previously. The consequences may be mitigated using gradient clipping heuristics, which reduces the step size to be small enough.

Remark 2.3.42. *Challenges: Long-Term Dependencies*
Occurs when computational graph becomes very deep. Let a computational graph contain a path that repeatedly multiply a matrix $\boldsymbol{W}$. After t steps, this is $\boldsymbol{W}^t$. Let the eigen-decomposition of $\boldsymbol{W}$ be $\boldsymbol{W} = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}$. Then we can see that $\boldsymbol{W}^t = (\boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1})^t = \boldsymbol{V}\text{diag}(\lambda)^t\boldsymbol{V}^{-1}$.
There will be a vanishing and exploding gradient problem if gradients on the graph are scaled according to $\text{diag}(\boldsymbol{\lambda})^t$. Note that recurrent networks uses same $\boldsymbol{W}$ at each time step, hence may face the problem.

Remark 2.3.43. *Challenges: Inexact Gradients*
Objective function for minimisation is intractable due to noisy/biased estimate of these quantities; in this case its gradient is intractable as well. Choose surrogate loss function that is easier to approximate than true loss.

Remark 2.3.44. *Challenges: Poor Correspondence between Local and Global Structure*
As gradient descent algorithms make small local moves, optimisation on local downhill moves may fail if local surface does not point toward the global solution. In higher dimensional space, learning algorithms may incur high costs or long training time to circumnavigate the manifold. Hence to find good initial points for problems.

Remark 2.3.45. *Stochastic Gradient Descent (SGD)*
Crucial parameter is the learning rate, which may be changed over time; at iteration k this is ϵ_k.
The SGD gradient estimation introduces a source of noise that does not vanish even at minimum. A sufficient condition to guarantee converge of SGD is

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \quad \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

In practice, it is common to decay learning rate linearly until iteration τ:

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau, \quad \alpha = \frac{k}{\tau}$$

After iteration τ, leave ϵ constant.
Compute time per update does not growth with number of training examples.

Algorithm 3: Stochastic Gradient Descent (SGD) Update at Training Iteration k

```
Require:
  Learning rate $\epsilon_k$, Initial parameter $\boldsymbol{\theta}$

  while stopping criterion not met do
    Sample a minibatch of $m$ samples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with targets $\boldsymbol{y}^{(i)}$
    Compute gradient estimate $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
  end while
```

Remark 2.3.46. *Momentum Algorithm*
Algorithm introduces variable $\boldsymbol{v}$ that is set to an exponentially decaying average of negative gradient.
Hyperparameter $\alpha \in [0, 1)$ determines how quickly the contributions of previous gradients exponentially decay.
The update rule is given by

$$\begin{aligned}\boldsymbol{v} &\leftarrow \alpha\boldsymbol{v} - \epsilon\nabla_{\boldsymbol{\theta}}\left(\frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})\right) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{v}\end{aligned}$$

The larger α is relative to ϵ, the more previous gradients affect current direction.
If gradient is $\boldsymbol{g}$, it will accelerate in direction of $-\boldsymbol{g}$ until reaching terminal velocity where size of each step is

$$\frac{\epsilon\,\|\boldsymbol{g}\|}{1-\alpha}$$

Algorithm 4: Stochastic Gradient Descent (SGD) with Momentum

```
Require:
  Learning rate $\epsilon_k$, Momentum parameter $\alpha$

  while stopping criterion not met do
    Sample a minibatch of $m$ samples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with targets $\boldsymbol{y}^{(i)}$
    Compute gradient estimate $\boldsymbol{g} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

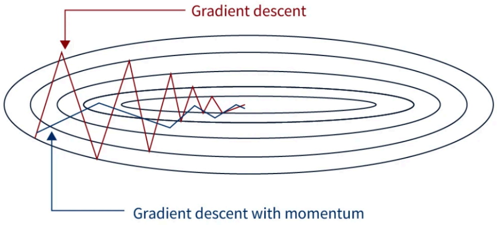

Figure 7: Stochastic Gradient Descent (SGD) Algorithm with and without Momentum

Remark 2.3.47. *Nesterov Momentum*
Inspired by Nesterov's accelerated gradient method, the update rules are:

$$\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\nabla_{\boldsymbol{\theta}}\left(\frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha\boldsymbol{v}), \boldsymbol{y}^{(i)})\right)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$

Note, gradient is evaluated after current velocity is applied.
Hence, Nesterov momentum is adding a correction factor to the standard method of momentum.

Algorithm 5: Stochastic Gradient Descent (SGD) with Nesterov Momentum

```
Require:
  Learning rate $\epsilon_k$, Momentum parameter $\alpha$
  Initial parameter $\boldsymbol{\theta}$, Initial velocity $\boldsymbol{v}$

  while stopping criterion not met do
    Sample a minibatch of $m$ samples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with targets $\boldsymbol{y}^{(i)}$
    Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{v}$
    Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

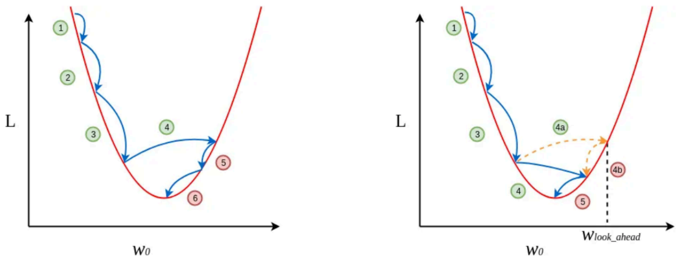

Figure 8: Momentum vs Nesterov Gradient Descent

Remark 2.3.48. *Parameter Initialisation Challenges*
Training deep neural networks is difficult as most algorithms are strongly affected by choice of initialisation.

Designing initialisation strategies is difficult as neural network optimisation is not yet well understood, as the nice properties from initialisation may not be preserved as training proceeds.
Some initial points may be beneficial for optimisation, but detrimental for generalisation.

Remark 2.3.49. *Parameter Initialisation Strategies*
Initial parameters need to 'break symmetry' between different units. May explicitly search for large set of basis functions mutually different from each other, but this may incur noticeable computational cost.
Random initialisation from high-entropy distribution over high-dimensional space is computationally cheaper.
Larger initial weights will yield stronger symmetry breaking effect and avoid redundant units. However, if weights are set too large, may result in chaos (extreme sensitivity to small perturbations of input). Exploding gradient may be mitigated by gradient clipping. The bias (i.e., parameters encoding conditional variance of a prediction) for each unit is set to heuristically chosen constraints.

Remark 2.3.50. *Weight Initialisation Heuristics*
Initialising parameters of neural network $\boldsymbol{\theta}$ to $\boldsymbol{\theta}_0$ is similar to imposing a Gaussian prior $p(\boldsymbol{\theta})$ with mean $\boldsymbol{\theta}_0$, hence it make sense to choose $\boldsymbol{\theta}_0$ near 0. Initialisation of $\boldsymbol{\theta}_0$ to large values means prior specifies which units should interact with each other, and how they should interact.
For a fully connected neural network with m inputs and n outputs, weights may be initialised by:

i. Sampling from $U\left(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\right)$

ii. Sampling from normalised initialisation $W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$ (Glorot and Bengio). Compromise between goal of initialising all layers to have same activation variance and that of same gradient variance. Derived from assumption that network is chain of matrix multiplications with no nonlinearities.

iii. Sampling from random orthogonal matrices, with scaling or gain factor g that accounts for nonlinearity at each layer, with specific values for different types of nonlinear activation functions. Assumes neural network is sequence of matrix multiplications without nonlinearities. Guarantees the total number of training iterations required to reach convergence is independent of depth.
Increasing scaling factor g pushes network toward regime where activations increase in norm in forward propagation, and gradients increase in norm in backward propagation.
For scaling rules that set all of initial weights to have same standard deviation, individual weight becomes extremely small when layers become large.

iv. Sparse initialisation (Martens), where each unit is initialised to have exactly k non-zero weight. Keep total number of input to unit independent from number of inputs m without making magnitude of individual weight elements shrink with m.
Imposes very strong prior on weights that are chosen to have large Gaussian values.

If computational resources allow, treat initial scale of weights for each layer as a hyper-parameter, and use hyper-parameter search algorithm. Choice of dense or sparse initialisation may also be a hyper-parameter.

Remark 2.3.51. *Bias Initialisation Heuristics*
Setting bias to zero is compatible with most weight initialisation schemes. The few other situations are:

i. If bias is for output unit, initialise bias to obtain right marginal statistics of the output. Assume initial weights are small enough that output of unit is determined only by bias. The bias will be inverse of activation function applied to marginal statistics of the output in training set.
If output is a highly skewed distribution over classes with marginal probability of class i given by element c_i of some vector $\boldsymbol{c}$, then bias vector $\boldsymbol{b}$ is such that softmax$(\boldsymbol{b}) = \boldsymbol{c}$. The models have output layers that should resemble input data $\boldsymbol{x}$, hence bias should match marginal distribution over $\boldsymbol{x}$.

ii. If bias is to be set to avoid causing too much saturation at initialisation (i.e., with ReLU). Not compatible with weight initialisation schemes that do not expect strong input form bias (i.e., with random walk init).

iii. If a unit controls whether other units are able to participate in a function. There is unit with output u and another unit $h \in [0, 1]$, multiplied together to produce output uh. Then h is a gate that determines if $uh \approx u$ or $uh \approx 0$. Hence, to set bias for h such that $h \approx 1$ at initialisation, else u does not learn.

Remark 2.3.52. *Variance or Precision Parameter Initialisation Heuristics*
For linear regression with conditional variance estimate using model

$$p(y \mid \boldsymbol{x}) = \mathcal{N}(y \mid \boldsymbol{w}^T \boldsymbol{x} + b, 1/\beta)$$

where β is a precision parameter. The variance of precision parameters may be initialised to 1.
Another approach is to assume initial weights are close enough to zero, and bias may be set while ignoring effect of the weights, then set bias to produce the correct marginal mean of the output, and set variance parameters to the marginal variance of the output in the training set.

Remark 2.3.53. *Model Parameter Initialisation with Machine Learning*
Initialise supervised model with parameters learned by unsupervised model trained on same inputs. Alternatively, perform supervised training on a related task, which may yield initialisation that offers faster convergence than random initialisation. Information on distribution is encoded into initial parameters of model.

Remark 2.3.54. *Adaptive Gradient (AdaGrad) Algorithm*
Algorithm individually adapts learning rates of all model parameters by scaling them inversely proportional to square root of sum of all of their historical squared values. Has greater progress in more gently sloped directions of parameter space. On deep neural networks, accumulation of squared gradients from beginning of training can result in premature and excessive decrease in effective learning rate.

Algorithm 6: Adaptive Gradient (AdaGrad) Algorithm

```
Require:
  Global Learning rate $\epsilon$
  Initial parameter $\boldsymbol{\theta}$
  Small constant $\delta$, perhaps $10^{-7}$ for numerical stability

  Initialise gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
     Sample a minibatch of $m$ samples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with targets $\boldsymbol{y}^{(i)}$
     Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
     Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
     Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$ (Division and square root applied element-wise)
     Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

Remark 2.3.55. *Root Mean Squared Propagation (RMSProp) Algorithm*
Modifies AdaGrad to perform better in non-convex setting by changing gradient accumulation into exponentially weighted moving average, which discards history from extreme past so it can converge rapidly after finding a convex bowl. Use of moving average introduces a new hyper-parameter ρ that controls length scale of the moving average. RMSProp is effective and practical for deep neural networks.

Algorithm 7: Root Mean Squared Propagation (RMSProp) Algorithm

```
Require:
  Global Learning rate $\epsilon$
  Decay rate $\rho$
  Initial parameter $\boldsymbol{\theta}$
  Small constant $\delta$, perhaps $10^{-6}$ to stabilise division by small numbers

  Initialise accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
     Sample a minibatch of $m$ samples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with targets $\boldsymbol{y}^{(i)}$
     Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
     Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho\boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$
     Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$ ($\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
     Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

Algorithm 8: Root Mean Squared Propagation (RMSProp) Algorithm

```
Require:
  Global Learning rate $\epsilon$
  Decay rate $\rho$
  Momentum coefficient $\alpha$
  Initial parameter $\boldsymbol{\theta}$
  Initial velocity $\boldsymbol{v}$

  Initialise accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
     Sample a minibatch of $m$ samples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with targets $\boldsymbol{y}^{(i)}$
     Compute interim update $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{v}$
     Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
```

```
        Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$
        Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$ ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
        Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
    end while
```

Remark 2.3.56. *Adaptive Moment Estimation (Adam) Algorithm*
Variant on combination of RMSProp and momentum. Note, momentum is incorporated directly as an estimate of first order moment of gradient; bias corrections are made to estimates of both first-order moments and second-order moments to account for initialisation at origin.
Algorithm is fairly robust to choice of hyper-parameters.

Algorithm 9: Adaptive Moment Estimation (Adam) Algorithm

```
Require:
    Step size $\epsilon$ (Suggested default: 0.001)
    Exponential decay rates for moment estimates $\rho_1, \rho_2$ in $[0, 1)$ (Suggested defaults: 0.9 and 0.999)
    Small constant $\delta$ used for numerical stabilisation (Suggested default: $10^{-8}$)
    Initial parameters $\boldsymbol{\theta}$

    Initialise 1st and 2nd momentum variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
    Initialise time step $t = 0$
    while stopping criterion not met do
        Sample a minibatch of $m$ samples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with targets $\boldsymbol{y}^{(i)}$
        Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
        Update time step $t \leftarrow t + 1$
        Update biased first and second moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g}$ and $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1-\rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
        Correct bias in first and second moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$ and $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
        Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}+\delta}}$ (Operations applied element-wise)
        Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
    end while
```

Remark 2.3.57. *Newton's Method*
Based on second-order Taylor's series approximating $J(\boldsymbol{\theta})$ near the point $\boldsymbol{\theta}_0$, ignoring higher derivatives

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{H} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

where $\boldsymbol{H}$ is the Hessian of J with respect to $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}_0$.
Solving for critical point will get the Newton parameter update rule

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Note that for deep learning, the surface of objective function is non-convex. Regularising the Hessian will solve many of the issues such as saddle points etc.

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - [H(f(\boldsymbol{\theta}_0)) + \alpha \boldsymbol{I}]^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Due to requirement for inversion of the $k \times k$ matrix with time complexity $O(k^3)$ at every training iteration, only networks with very small number of parameters may be trained with this method.

Algorithm 10: Newton's Method with Objective $J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$

```
Require:
    Initial parameter $\boldsymbol{\theta}_0$
    Training set of $m$ examples

    while stopping criterion not met do
        Compute gradient $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
        Compute Hessian: $\boldsymbol{H} \leftarrow \frac{1}{m} \nabla^2_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
        Compute Hessian Inverse: $\boldsymbol{H}^{-1}$
        Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$
        Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
    end while
```

Remark 2.3.58. *Conjugate Gradient Method*
Method effectively avoids computation of inversion Hessian by descending conjugate directions, which is a search direction conjugate to the previous line search direction (will not undo progress made in that direction).
Let current and previous search directions be $\boldsymbol{d}_t, \boldsymbol{d}_{t-1}$. At training iteration t, the next search direction $\boldsymbol{d}_t$ is

$$\boldsymbol{d}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \beta_t \boldsymbol{d}_{t-1}$$

note that β_t may be computed via Fletcher Reeves:

$$\beta_t = \frac{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$$

or may be computed with Polak-Ribiere:

$$\beta_t = \frac{(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1}))^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$$

The non-linear conjugate gradient method is a variation which uses occasional resets where the method is restarted with line search along with the unaltered gradient.

Algorithm 11: Conjugate Gradient Method

```
Require:
  Initial parameter $\boldsymbol{\theta}_0$
  Training set of $m$ examples
  Initialise $\boldsymbol{\rho}_0 = 0$
  Initialise $g_0 = 0$
  Initialise $t = 1$

  while stopping criterion not met do
      Initialise the gradient $\boldsymbol{g}_t = \boldsymbol{0}$
      Compute gradient $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
      Compute Polak-Ribiere metric: $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^T \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^T \boldsymbol{g}_{t-1}}$
      (Nonlinear conjugate gradient: optionally reset $\beta_t = 0$, i.e., if $t$ is a multiple of some constant $k$)
      Compute search direction: $\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$
      Perform line search to find: $\epsilon^* = \arg\min_\epsilon \frac{1}{m} \sum_{i=1}^m L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_t + \epsilon \boldsymbol{\rho}_t), \boldsymbol{y}^{(i)})$
      (On truly quadratic cost function, analytically solve for $\epsilon^*$)
      Apply update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^\theta \boldsymbol{\rho}_t$
      Apply time step $t \leftarrow t + 1$
  end while
```

Remark 2.3.59. *Nonlinear Conjugate Gradients*
Method of conjugate gradients is still applicable for non-linear objective function.
Conjugate directions are no longer assumed to remain at minimum of the objective for previous directions. Has occasional resets where method of conjugate gradients is restarted with line search along unaltered gradient.

Remark 2.3.60. *Broyden-Fletcher-Goldfarb-Shanno (BGFS) Algorithm*
Similar to conjugate gradient, but uses more direct approach to approximation of newton's update.
Method approximate the inverse of hessian $\boldsymbol{H}$ with matrix $\boldsymbol{M}_t$, iteratively refined by low rank updates to better approximate $\boldsymbol{H}^{-1}$. Direction of gradient descent is then determined by $\boldsymbol{\rho}_t = \boldsymbol{M}_t \boldsymbol{g}_t$. Line search is performed in this direction to determine step size ϵ^*. Final update of parameter is then $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$.
Note, BGFS store matrix $\boldsymbol{M}$ which requires $O(n^2)$ memory, making it impractical for most modern models.

Remark 2.3.61. *Limited Memory BFGS (L-BFGS) Algorithm*
Memory costs of BFGS may be decreased by not storing complete inverse Hessian approximation $\boldsymbol{M}$. L-BFGS computes approximation $\boldsymbol{M}$ with same method as BFGS, but assumes at beginning that $\boldsymbol{M}^{(t-1)}$ is the identity matrix, rather than storing the approximation between steps.
If used with exact line searches, directions defined by L-BFGS are mutually conjugate, and this procedure remains well-behaved when minimum of line search is reached only approximately.
L-BFGS may be generalised to store some vectors used to update $\boldsymbol{M}$ at each time step, costing $O(n)$ per step.

Remark 2.3.62. *Meta-Algorithm: Batch Normalisation*
An update from optimisation algorithm may result in multiple-order effects, and in very deep networks, even higher-order interactions can be significant.

Batch normalisation provides an elegant way of re-parametrising any deep network, which reduces the problem of coordinating updates across multiple layers. Batch normalisation may be applied to any input/hidden layer. Let $\boldsymbol{H}$ be minibatch of activations of layer to normalise, with activations for each example appearing in a row. To normalise $\boldsymbol{H}$, replace it with $\boldsymbol{H}' = \frac{\boldsymbol{H}-\boldsymbol{\mu}}{\boldsymbol{\sigma}}$ where $\boldsymbol{\mu}$ is a vector of mean of each unit and σ is a vector of standard deviation of each unit. Then H_{ij} is normalised by subtracting μ_j and dividing σ_j. Rest of network then operates on $\boldsymbol{H}'$ just as in original network on $\boldsymbol{H}$.
At training, the mean and standard deviations are computed as

$$\boldsymbol{\mu} = \frac{1}{m}\sum_i \boldsymbol{H}_{i,:}, \quad \boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m}\sum_i (\boldsymbol{H}-\boldsymbol{\mu})_i^2}$$

where δ is a small positive value (i.e., 10^{-8}) to avoid undefined gradient of $\sqrt{z}$ at $z = 0$.
These are then back-propagated to normalise $\boldsymbol{H}$, hence gradient will never increase the standard deviation or mean of h_i. Batch normalisation re-parametrises the model to make some units always standardised.
At test time, μ and σ may be replaced by running averages from training time. This allows model to be evaluated on a single example, without needing to use definitions of μ and $\boldsymbol{\sigma}$ that depends on entire minibatch.

Remark 2.3.63. *Batch Normalisation but Maintain Network Express Power*
Replace batch of hidden activations $\boldsymbol{H}$ with $\boldsymbol{\gamma H}' + \boldsymbol{\beta}$ rather than $\boldsymbol{H}'$, where $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learned parameters to allow new variable to have any mean and standard deviation. With original $\boldsymbol{H}'$, mean of $\boldsymbol{H}$ was determined by complicated interaction between parameters in layers below $\boldsymbol{H}$. With $\boldsymbol{\gamma H}' + \boldsymbol{\beta}$, the mean is determined solely by $\boldsymbol{\beta}$, which is much easier to learn with gradient descent.
Most neural network layers are of form $\phi(\boldsymbol{XW}+\boldsymbol{b})$, where ϕ is some fixed nonlinear activation function. Then $\boldsymbol{XW}+\boldsymbol{b}$) should be replaced by normalised version of $\boldsymbol{XW}$, and bias term omitted as it becomes redundant with β parameter applied from batch normalisation re-parametrisation. Statistics of input are non-Gaussian and less amenable to standardisation by linear operations.

Remark 2.3.64. *Meta-Algorithm: Coordinate Descent*
Solving optimisation problems by minimising the objective function $f(\boldsymbol{x})$ with respect to a single variable x_i, and rapidly cycling through all variables. *Block coordinate descent* refers to minimising with respect to a subset of variables simultaneously.
Coordinate descent works when different variables can be clearly separated into groups that play relatively isolated roles, or when optimisation with respect to one group of variables is significantly more efficient than optimisation with respect to all of the variables.

Example 2.3.65. *Sparse Encoding with Coordinate Descent*
Consider the cost function

$$J(\boldsymbol{H},\boldsymbol{W}) = \sum_{i,j}|H_{ij}| + \sum_{i,j}(\boldsymbol{X} - \boldsymbol{W}^T\boldsymbol{H})_{i,j}^2$$

To find weight matrix $\boldsymbol{W}$ that can linearly decode a matrix of activation values $\boldsymbol{H}$ to reconstruct training set $\boldsymbol{X}$. Most applications also include weight decay or constraint on norms of columns of $\boldsymbol{W}$ to prevent the pathological solution with extremely small $\boldsymbol{H}$ and large $\boldsymbol{W}$.
Note J is not convex, but inputs may be divided into the dictionary parameters $\boldsymbol{W}$ and the code representations $\boldsymbol{H}$. Minimising object function with respect to either set is a convex problem.

Remark 2.3.66. *Polyak Averaging*
Average several points in trajectory through parameter space visited by optimisation algorithm.
If t iterations of gradient descent visit points $\boldsymbol{\theta}^{(1)},\ldots,\boldsymbol{\theta}^{(t)}$, then the output is $\hat{\boldsymbol{\theta}}^{(t)} = \frac{1}{t}\sum_i \boldsymbol{\theta}^{(i)}$.
The optimisation algorithm may leap back and forth across a valley several times without visiting the minimum point, and the average of all points on either side of the valley should be close to minimum value.
This is developed solely for stochastic gradient descent (SGD) to stabilise the estimates (Granziol et al. (2020)).
For non-convex problems, an exponentially decaying running average is used:

$$\hat{\boldsymbol{\theta}}^{(t)} = \alpha\hat{\boldsymbol{\theta}}^{(t-1)} + (1-\alpha)\boldsymbol{\theta}^{(t)}$$

Remark 2.3.67. *Supervised Pre-training*
Pre-training refers to training simple models on simple tasks before training the desired model on desired task. Greedy algorithms break problem into many components, then solve for optimal version of each component in isolation, which is must cheaper than algorithms that solve for best joint solution. Fine-tuning on joint optimisation algorithm searches for optimal solution to the full problem.
Greedy supervised training breaks supervised learning problems into other simpler supervised learning problems.

Transfer learning refers to pre-training a deep network on a set of tasks, then initialise a same-size network with first k layers of first net. The layers of second networks (upper layers initialised randomly) are then jointly trained to perform a different set of tasks with fewer training examples than for first set of tasks.
FitNets approach begins by training a network with low depth and high width, which becomes a teacher for a second network (the student), which is much deeper and thinner. Student network is trained to not only predict the output on original task, but also on value of middle layer of teacher network.

Remark 2.3.68. *Neural Network Design Choice: Linear Transformation*
Model units have generally moved towards using more linear functions, which allow model to have nice properties that make optimisation easier. The model's local gradient information corresponds reasonably well to moving towards a distant solution.

Remark 2.3.69. *Neural Network Design Choice: Linear Paths and Skip Connections*
Linear paths and skip connections between layers reduce length of shortest path from lower layer's parameters to the output, mitigating vanishing gradient problem.
Adding extra copies of output that are attached to intermediate hidden layers is an alternative to pre-training, where these 'auxiliary heads' are trained to perform the same task as primary output at the top of the network to ensure that the lower layers receive a large gradient. After training, these may be discarded.

Remark 2.3.70. *Continuation Methods*
Family of strategies that can make optimisation easier by choosing initial points to ensure that local optimisation spends most of its time in well-behaved regions. Method constructs series of objective functions $\{J^{(0)}, \dots, J^{(n)}\}$ which are increasingly difficult to minimise (well-behaved over less regions of $\boldsymbol{\theta}$ space) for the true cost function $J(\boldsymbol{\theta})$. The cost functions are designed such that a solution to one is a good initial point for the next.
Continuation methods aim to reach a global minimum despite many local minima. Easier cost functions are constructed through the blurring operation, which approximates

$$J^{(i)}(\boldsymbol{\theta}) = \mathbb{E}_{\theta' \sim \mathcal{N}(\boldsymbol{\theta}'; \boldsymbol{\theta}, \sigma^{(i)2})} J(\boldsymbol{\theta})'$$

via sampling. Some non-convex functions become approximately convex when blurred, but preserved enough information about local global minimum. However, approach might not work when

i. so many incremental cost functions are required that cost of entire procedure remains high;
ii. NP-hard optimisation problems remain NP-hard, even when continuation methods are applicable;
iii. function might not become convex, no matter how much it is blurred;
iv. function may become convex due to blurring, but minimum of this blurred function may track a local rather than global minimum of original cost function.

Remark 2.3.71. *Curriculum Learning*
Planning a learning process to begin by learning simple concepts, and progress to learning more complex topics that depend on these simpler concepts. Earlier objective functions $J^{(i)}$ are made easier by increasing the influence of simpler examples.
Stochastic curriculum is a variation of the approach where a random mix of easy and difficult examples is always presented to the learner, but average proportion of more difficult examples is gradually increased.

2.3.4 Convolutional Neural Networks

Definition 2.3.72. *Convolutional Neural Networks (CNNs)*
Neural network for processing data with a known, grid-like topology. Uses convolution instead of general matrix multiplication in at least one of their layers.

Definition 2.3.73. *Convolution*
The convolution of two functions f and g, denoted by $*$, is the integral of product of two functions after one is reflected about the y-axis and shifted.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

The first argument $f(\tau)$ is the input, and second argument $g(t-\tau)$ is the kernel. The output is feature map.

Definition 2.3.74. *Discrete Convolution*
For complex-valued functions f and g defined on set of $\mathbb{Z}$ integers, the discrete convolution of f and g is

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

If function g_N is periodic with period N, then the convolution is also periodic and identical to

$$(f * g_N)[n] = \sum_{m=0}^{N-1} \left(\sum_{k=-\infty}^{\infty} f[m+kN] \right) g_N[n-m]$$

Note that convolution is commutative, as the kernel is flipped relative to the input (as m increases, the index into input increases, but index into kernel decreases), but this is not an important property of neural network implementation. Neural network libraries implement cross-correlation instead.

Definition 2.3.75. *Cross-Correlation*
A measure of similarity of two series as a function of the displacement of one relative to the other. Also known as *sliding inner or dot product*. This is defined as

$$(f \star g)(\tau) = \int_{-\infty}^{\infty} \overline{f(\tau)} g(t+\tau)dt$$

This is related to convolution by

$$[f(t) \star gt](t) = [\overline{f(-t)} * g(t)](t)$$

Remark 2.3.76. *Motivation: Sparse Interactions*
The kernel is smaller than input, but is able to detect meaningful features. Hence fewer parameters are to be stored, reducing memory requirements of the model. Units in deeper layers may indirectly interact with a larger portion of the input, hence efficiently describe complicated interactions between many variables.

Remark 2.3.77. *Motivation: Parameter Sharing*
Each member of kernel is used at every position of the input except at boundary, which depends on design decisions regarding the boundary. Only one set of parameter is required to be learned. The runtime of forward propagation is at $O(k \times n)$ for $k \lll m$ connections for each input, given m input and n outputs.

Remark 2.3.78. *Motivation: Equivariant Representations*
An *equivariant* function is such that if the input changes, the output changes in the same way.
Applying a transformation, then convolution, is equivalent to applying convolution, then the transformation.

Remark 2.3.79. *Typical CNN Layers*
The layers of a typical CNN are:

i. Convolution: layer performs several convolutions in parallel to produce set of linear activations

ii. Detector: each linear activation is run through a non-linear activation function

iii. Pooling: modify the output of the layer further with summary statistics of nearby outputs

Example 2.3.80. *Example Types of Pooling*

i. Max Pooling: reports maximum output for each region of the feature map

ii. Average Pooling: reports average for each region of the feature map

iii. L^2 Pooling: calculates Euclidean norm (L^2 norm) for each region of the feature map

Remark 2.3.81. *Pooling on Invariance*
Pooling makes the representation approximately invariant to small translations of input.
Use of pooling is analogous to adding an infinitely strong prior that the function the layer learns must be invariant to small translations. As pooling summarises responses over whole neighbourhood, fewer pooling units than detector units may be used. To handle inputs of varying size, the size of offset between pooling regions may be changed so that the classification layer always receives the same number of summary statistics.

Remark 2.3.82. *Multi-Channel Operations*
Not commutative unless each operation has the same number of output and input channels.
Assume there exists 4-D kernel tensor $\mathcal{K}$ with element $K_{i,j,k,l}$ giving connection between strength between a unit in channel i of the output and a unit in channel j of the input, with offset of k rows and l columns between the output unit and input unit. Assume input consists of observed data $\mathcal{V}$ with element $V_{i,j,k}$ giving the value of input unit within channel i at row j and column k. Assume output consists of $\mathcal{Z}$ in same format as $\mathcal{V}$. If $\mathbb{Z}$ is produced by convolving $\mathcal{K}$ across $\mathcal{V}$ without flipping $\mathcal{K}$, then

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

where summation over l, m, n is over all values for which tensor indexing operations inside summation is valid.

Remark 2.3.83. *Downsampled Convolution*
To reduce computational cost by sampling only every s pixels (*stride*) in each direction of output, then

$$Z_{i,j,k} = c(\mathcal{K}, \mathcal{V}, s)_{i,j,k} = \sum_{l,m,n} [\mathcal{V}_{l,(j-1)\times s+m,(k-1)\times s+n} \mathcal{K}_{i,l,m,n}]$$

Remark 2.3.84. *Zero Padding*
Zero-padding is adding zeroes to end of input sequence $\mathcal{V}$ so that the total number of samples is equal to the next higher multiple of two.

i. Valid Convolution: no zero-padding used. If input image has width m and kernel has width k, the output will be width $m - k + 1$. Rate of shrinkage can be dramatic if kernel used are large.

ii. Same Convolution: just enough zero-padding is added to keep size of output equal to size of input. Border pixels may be underrepresented in the model.

iii. Full Convolution: zeroes are added for every pixel to be visited k times in each direction, resulting in an input image of width $m + k - 1$.

Remark 2.3.85. *Locally Connected Layers (Unshared Convolution)*
Similar operation to discrete convolution with small kernel, but without parameter sharing. Adjacency matrix of MLP is the same, but every connection has its own weight as specified by 6-D tensor $\mathcal{W}$, where the indices of $\mathcal{W}$ are the output channel i, the output row j, the output column k, the input channel l, the row offset within the the input m, and the column offset within the input n. The liner part of the layer is then

$$Z_{i,j,k} = \sum_{l,m,n} [\mathcal{V}_{l,j+m-1,k+n-1} w_{i,j,k,l,m,n}]$$

Locally connected layers are useful when it is known that each feature should be function of a small part of space but would not occur across all of space.

Remark 2.3.86. *Tiled Convolution*
Compromise between convolutional layer and locally connected layer. A set of kernels will be rotated through space. Immediate neighbouring locations will have different filters, but memory requirements for storing the parameters will increase only by a factor of the size of this set of kernels.
Let k be a 6-D tensor, where two of the dimensions correspond to different locations in output matrix. Output locations cycle through a set of t different choices of kernel stack in each direction.

$$Z_{i,j,k} = \sum_{l,m,n} \mathcal{V}_{l,j+m-1,k+n-1} \mathcal{K}_{i,l,m,n,j \bmod t+1,k \bmod t+1}$$

3 Research

3.1 Research in AI

3.1.1 Reading AI Papers

Remark 3.1.1. *Reading Wide*
Navigate through literature reading small amounts of individual research papers, build and improve mental model of research topic.
On *Papers with Code*, check out top benchmark models, read abstract and take note of key information only. Next step for each model, look into the datasets used by the papers and make notes.
On *Google Scholar*, check if there is any survey papers (reviews and describe state of problem space with challenges and opportunities) to get up to speed, read each paper, and make notes.
Next find related works (recently published) to understand how researchers in the field traditionally approached the problems, and what the emerging trends are. Related works will populate reading list.

Remark 3.1.2. *Reading Deep*
First pass will not understand more than 10% of research paper, and may require reading of another more fundamental paper. Then read subsequent passes until 70% $\sim$ 80% understanding.
In introduction, highlight problems and challenges, solutions to challenges, main contributions of the work. Should be able to extract a problem-solution chain from the introduction and proposed solution.
In methods, maintain list of concepts not yet understood. If there is link to paper references, keep track.
In experiments, highlight data setup, main table of results. Keep track of unfamiliar evaluation metrics.

Remark 3.1.3. *Practical AI Research Tools*
Experiment tracking tools include 'Weights & Biases', 'Tensorboard', 'Neptune'.
If using 'Weights & Biases', model artefacts can be stored directly on a system of choice.
To train deep learning pipelines by using a config which an modify depending on which dataset, model, or configuration is used, Python's Hydra package may be used.

Remark 3.1.4. *Identifying Gaps in Research Paper*

i. Identify gaps in research question by comparing with the research hypotheses of compiled papers

ii. Identify gaps in experimental gaps, such as shortcomings in evaluation of methods, the way the comparisons were chosen or implemented, and whether the experimental setup tests the research hypothesis decisively.

iii. Identify gaps through expressed limitations, implicit and explicit.

3.1.2 Writing AI Papers

Remark 3.1.5. *Generating Ideas for Building on Research Paper*

i. Change task of interest. Can the main ideas be applied to a different modality, a different data type? Can the method or learned model be applied to a different task? Can the outcome of interest be changed?

ii. Change the evaluation strategy. Can it be evaluated on a different dataset or different metric? Explore why something works well/breaks. Make different comparisons.

iii. Change the proposed method. Can training dataset or data elements be changed? Can the pre-training/training strategy be changed? Can the deep learning architecture or problem fomulation be changed?

Remark 3.1.6. *Iterating on Research Ideas*

i. Search for whether idea has been tried. Construct titles for new paper ideas, see if there is Google result.

ii. Read important related works and follow up works.

iii. Geed feedback from domain experts on drafted ideas in written form. Email to authors of the work being built on, share idea and plan, ask their opinion.

Remark 3.1.7. *Global Structure of ML Papers*
ML Papers follow the following pattern (in 6 to 7 sections):

i. Abstract (answers 5 to 6 canonical questions in $100 \sim 250$ words):

   1. What is the background and gap?
   2. What is the key desideratum?
   3. What is the proposed solution?
   4. What are its main components?
   5. What are its strengths?
   6. What are the notable results (with tasks, numbers)?

ii. Introduction (start and end position are nearly identical across papers):

   1. Context, success of prior approaches
   2. Weaknesses or gaps of prior methods
   3. Desiderata for an improved solution
   4. Proposed method: key components, contributions
   5. High-level experimental overview, positive results

iii. Related Work/Background (typically 2 to 3 subsections), each paragraph conveys:

   1. High-level mapping of approach categories
   2. Evolution of methods over time
   3. How the proposed method compares to each category
   4. What gaps persist

iv. Methods (content always includes):

   1. Overall approach description
   2. Architecture, input/output flow
   3. Loss functions, training objectives
   4. Implementation details
   5. (Sometimes) dataset descriptions or task-specific usage

v. Experiments (how results are conveyed. Always structured as such):

   1. Overall evaluation setup
   2. Dataset description (if not earlier)
   3. Implementation details
   4. Results per task type (tables/figures)
   5. Ablations at the end
   6. References to earlier figures and comparisons to prior models

vi. Conclusion, Broader Impacts (mirrors abstract but expands):

   1. Solution, components
   2. Strengths, notable results
   3. Future directions
   4. Limitations and societal considerations
   5. Motivations for follow-up work

Remark 3.1.8. *Figure Progression of ML Papers*
ML Papers follow the following pattern for figures:

i. Method overview diagram (always first)

ii. Lower-level architecture or objective illustration

iii. Comparisons vs. prior models across multiple tasks

iv. Ablation studies

v. (Optional) qualitative examples, predictions, dataset samples

References

Granziol, D., X. Wan, and S. Roberts (2020). Gadam: Combining adaptivity with iterate averaging gives greater generalisation. *arXiv preprint arXiv:2003.01247*.

Hamilton, J. D. (1994). *Time Series Analysis*. Princeton University Press.

Mazur, M. (2015). A step-by-step backpropagation example. Accessed: 2025-04-27.

Narang, R. K. (2013). *Inside the Black Box: The Simple Truth About Quantitative Trading*. Wiley Finance Series. Wiley.

Prado, M. L. D. (2018). *Advances in Financial Machine Learning*. Wiley Finance Series. Wiley.

Thierry, A. and G. Helyette (2000, October). Order flow, transaction clock, and normality of returns. *The Journal of Finance 55*(5), 2259–2284.

Velu, R. (2020). *Algorithmic Trading and Quantitative Strategies*. CRC Press.