KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Multimodal Tweet Sentiment for Implied Volatility Predictions

Arthur Decloedt

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotor:
Prof. dr. Jesse Davis, Prof. dr. Wim Schoutens

Assessor:
ir. Tom Decroos, dr. Nyi-Nyi Htun

Begeleider:
ir. Thomas Dierckx

Academiejaar 2019 – 2020

Preface

This thesis is the culmination of 6 long years at the KULeuven. I would like to thank my promotors Prof. Davis and Prof. Schoutens for providing ideas and direction to this project and mr. Dierckx for doing what he could to keep me headed in that direction. I would also like to thank my parents, friends and family for their support. *Some of the computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government.*

Arthur Decloedt

Contents

Abstract

This thesis researches the extraction of multimodal sentiment from tweets referencing an equity and approaches on using it for predicting future implied volatility on that equity. A scalable and efficient data collection procedure is defined and implemented. The constructed dataset contains both textual and visual information with a sub day level granularity. Emotional features from both modalities are extracted using pre-trained models with a decision level fusion approach, resulting in a dataset reflecting popular sentiment towards a specific equity over multiple years with sentiment vectors for each tweet. We present several lightweight, size agnostic architectures, derived from convolutional deep averaging networks, that outperform out of the box machine learning approaches. While the used dataset is relatively small and rough, the neural networks show promise in this task. We finally demonstrate the models outperforming baselines set by the lagged value when relaxing the constraint of only using the multimodal data.

Samenvatting

Introductie

Toekomstige waardes voorspellen is een belangrijk onderdeel van artificiële intelligentie en financiële marktvariabelen zijn een aantrekkelijke plaats om dergelijke voorspellingen te doen. Er zijn echter verschillende redenen waarom dit niet zo gemakkelijk is als soms verondersteld. Het accuraat voorspellen van toekomstige marktsituaties is vooral nuttig als de voorspeller hierin beter is dan de markt als geheel, wat ingaat tegen de Efficiënte Markt Hypothese [Malkiel, 2003a].

Een belangrijke financiële variabele is de geïmpliceerde volatiliteit (IV), die kan berekend worden op zowel indexen (bv de VIX op de S&P500) als op specifieke aandelen (bv VXAAPL op $AAPL). De geïmpliceerde volatiliteit indiceert de volatiliteit ingecalculeerd in de prijzen van opties over de volgende 30 dagen; ze kan berekend worden door het invers oplossen van een optie-prijs probleem. De VIX is een belangrijke variabele die soms de Angst-index wordt genoemd, aangezien ze stijgt bij onzekerheid. Het is echter belangrijk te onthouden dat die onzekerheid geen richting heeft.

Aangezien de volatiliteit sterk samenhangt met het sentiment van de markt lijkt het ons een interessant probleem om deze IV te voorspellen door middel van publiek sentiment, dat we uit verschillende soorten informatie in tweets zullen halen.

Sentiment Analyse

Het is voor mensen zeer natuurlijk om naar de gevoelens van andere mensen te peilen op basis van communicatie. Miljoenen jaren van evolutie heeft ons hier redelijk goed in gemaakt, maar hetzelfde geldt niet voor algoritmes. Sentiment analyses zijn dan ook een belangrijk onderzoeksveld in AI. Waar mensen vaak meerdere modaliteiten gebruiken om sentiment te bepalen zijn algoritmes vaak unimodaal, wat betekent dat ze zich specialiseren in een bepaalde soort informatie: meestal tekst maar ook afbeeldingen, geluid of video. Er zijn verschillende specifieke doelen voor een sentiment analyse. Men kan op zoek zijn naar een algemeen sentiment maar ook naar sentiment over een specifieke entiteit of een specifiek aspect van een entiteit. Men kan in plaats van simpele polariteit ook zoeken naar verschillende emoties, vaak uitgedrukt in de 24 dyades uit Plutchiks wiel van emoties [Plutchik, 1991].

Sentiment Uit Tekst Traditioneel is tekst de modaliteit met het meeste onderzoek op vlak van sentiment. Dit komt omdat tekst makkelijk voor te stellen is in datastructuren die een sterke link hebben met de betekenis, wat vitaal was voor de deterministische methodes die vroeger populair waren. De structuur en het functioneren van deze algoritmes waren volledig of grotendeels bedacht door mensen en ze vertoonden geen lerend gedrag. De simpelste methode is naar alle woorden in een tekst te kijken en er een waarde aan toe te kennen. De gemiddelde waarde nemen we als sentiment, dit noemt men een bag-of-words aanpak, zoals gedemonstreerd in [Mishne et al., 2005]. Taal is echter meer dan een hoop woorden, er zit veel syntactische informatie in zinnen, informatie die we vaak nodig hebben om de betekenis of het sentiment te bepalen. Dit uit zich bijvoorbeeld in negaties, verwijswoorden en woorden die meerdere betekenissen of implicaties hebben. Over het algemeen vereist het accuraat begrijpen van taal een diep inzicht in mensen, hoe ze denken, en de wereld waarin ze leven. Vroege algoritmes probeerden taal te begrijpen door op zoek te gaan naar features die door mensen waren opgesteld, via representaties en structuren expliciet door mensen gebouwd. Deze aanpakken worden tegenwoordig vaak overtroffen door moderne machine learning modellen, zeker in situaties met heel grote hoeveelheden data van redelijk goede kwaliteit. Dit is echter niet universeel het geval [Hutto and Gilbert, 2014].
Moderne methodes om sentiment uit tekst te halen zijn gebaseerd op moderne modellen in NLP en steunen vaak op heel grote en complexe neurale modellen, die volledig als een zwarte doos getraind worden. Meestal hebben ze ofwel een recurrente ofwel een autoregressieve structuur, de recente state of the art modellen zijn meestal van transformer architectuur [Devlin et al., 2018] [Raffel et al., 2019] [Dat Quoc Nguyen and Nguyen, 2020].

Sentiment Uit Afbeeldingen Afbeeldingen waren lange tijd een moeilijkere modaliteit voor algoritmes: ze hebben een grote dimensie, vertonen sterke spatialiteit en hebben een minder interpreteerbare structuur. Vroege computer vision algoritmes vereisten veel preprocessing om features uit de afbeeldingen te halen. Dit veranderde echter compleet door de combinatie van de GPU revolutie en convolutionele neurale netwerken (CNN) [Fukushima, 1988]. CNN's zijn gebaseerd op structuren gevonden in het menselijk brein die gebruikt worden bij het verwerken van visuele stimuli. Ze kunnen afbeeldingen als input aan zonder enige vorm van preprocessing. State of the art modellen gebruiken complexe connectie-schema's van convolutionele lagen [He et al., 2016] met een finale laag die de uiteindelijke voorspelling doet. Om snel en effectief sentiment te extraheren neemt men vaak een dergelijk voorgetraind netwerk en vervangt men de finale laag, dit noemt met meestal finetuning of transfer learning.

Multimodaal Leren

Het meeste onderzoek naar artificiële intelligentie gaat naar modellen die één soort informatie als input krijgen. Dit kan bijvoorbeeld tekst, geluid of afbeeldingen zijn. Deze soorten informatie noemen we modaliteiten. Mensen combineren echter vaak waarnemingen in verschillende modaliteiten. Ze combineren stimuli van sterk

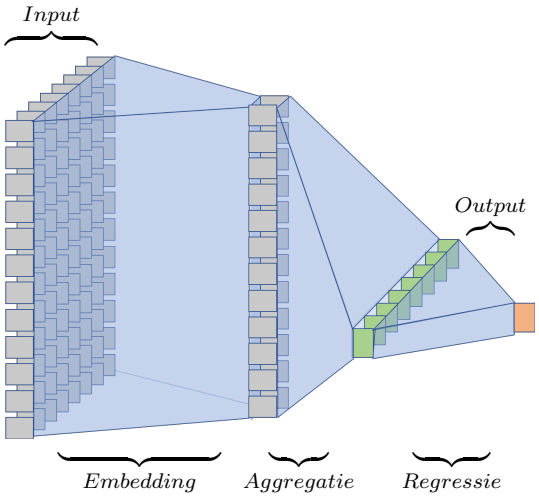
verschillende zintuigen die sterk verschillende prikkels waarnemen. In multimodaal leren trachten we sterk verschillende soorten informatie te combineren.

Traditionele Machine Learning Algoritmes

In de eerste experimenten zullen we twee eerder traditionele algoritmes gebruiken, zodat we later kunnen vergelijken met onze gespecialiseerde neurale netwerken. We kozen voor een simpele parametrische regressie en gradient boosted regressie bomen [Chen and Guestrin, 2016](XGBoost). Voor XGBoost is het nodig om de hyperparameters af te stellen. Dit zullen we doen met een Bayesiaanse optimalisatie, dit is een algoritme dat probeert een optimale waarde te vinden door iteratief punten te onderzoeken waarvan het denkt dat er opwaarts potentieel is.

Set Leren

Veel neurale modellen hebben vaste input en output dimensies. In sommige gevallen kan de input in grootte veranderen maar de output niet (tekst classificatie), in andere zijn de input en output beide van variabele grootte (machine vertaling). De output dimensie kan ten slotte ook een functie van de input dimensie zijn(vb U-nets, FCN's) De meest voorkomende architecturen voor variabele input groottes zijn recurrent of transformer gebaseerd. Hier gebruikt men interne staat of attentie (of een combinatie) om sequenties af te gaan. Dit soort netwerken heeft echter veel data nodig en werkt niet goed in de gevallen waar de input geen specifieke orde heeft, of waar er een serieus verschil is in de lengte van verschillende inputs [Hochreiter et al., 2001]. Een alternatief is een convolutioneel diep middelend netwerk (CDAN) [Gardner et al., 2019]. Hier worden convolutionele lagen gebruikt die over de variabele as van het netwerk bewegen, waarna een simpele aggregatie volgt. Tenslotte gebruikt men een traditioneel dens netwerk voor de uiteindelijke voorspelling.



Figuur 1: De 3 delen van een CDAN netwerk

Hypothese en Doelen

Dit project heeft twee hoofddoelen. Ten eerste zullen we een dataset samenstellen van een verzameling tweets, georganiseerd volgens datum, in 2 modaliteiten, visueel en tekstueel, om daar sentiment uit te halen. Ten tweede zullen we verschillende machine learning modellen gebruiken en evalueren om een voorspelling te doen van de geïmpliceerde volatiliteit op het aandeel voor de volgende dag.
We kunnen dit explicieter maken door te stellen dat we een dataset construeren D zodat

$$D : X \times Y \text{ waar } X : \mathcal{P}(\mathbb{R}^{embedding\,grootte}) \text{ en } Y : \mathbb{R}$$

Dit kan ook uitgedrukt worden als:

$$D = (x, y) = (x, f(y)) \text{ waar } f \in G : X \to Y$$

Hier is G de ruimte van functies, en f de grond waarheid die we willen leren.
Dan is ons doel om f^* te leren zodat

$$f \equiv f^*$$

Om f te benaderen zullen we de fout proberen minimaliseren

$$\epsilon = \int_X C(f(x), f^*(x))\, dx \text{ waar } C : Y \times Y \to \mathbb{R}$$

We hebben natuurlijk enkel de dataset, niet de volledige functie dus we zullen evalueren op een validatieset V.

$$\epsilon = \sum_V C(f(x), f^*(x))$$

Evaluatie van Predictieve Modellen

Voor onze dataset is het moeilijk om de kwaliteit kwantitatief te meten. Voor onze experimenten om waardes te voorspellen is dit echter explicieter doenbaar. Hier bespreken we de evaluatiemethode die we gebruiken om verschillende modellen te vergelijken.

Evaluatie Criteria

Om de prestaties van verschillende predictieve modellen te meten zullen we specifiek evalueren op een paar criteria: RMSE, MSE en MAE.

(Root) Mean Squared Error Deze metrieken indiceren de kwadratische afstand tussen de voorspelling en de eigenlijke waarheid. Doordat ze het gemiddelde van kwadraten nemen, zullen ze veel belang hechten aan uitschieters. RMSE is simpelweg de vierkantswortel van de MSE, en wordt getoond omdat hij in dezelfde schaal is als

de response variabele en de MAE. De MSE en RMSE over een validatieset V worden berekend als volgt:

$$\begin{aligned} MSE_V &= \frac{1}{\#V} \sum_V \left(f(x) - f^*(x)\right)^2 \\ RMSE_V &= \sqrt{\frac{1}{\#V} \sum_V \left(f(x) - f^*(x)\right)^2} \end{aligned} \tag{1}$$

MSE wordt vaak gebruikt als een loss-functie voor gradiënt gestuurde optimalisaties zoals XGBoost maar ook SGD.

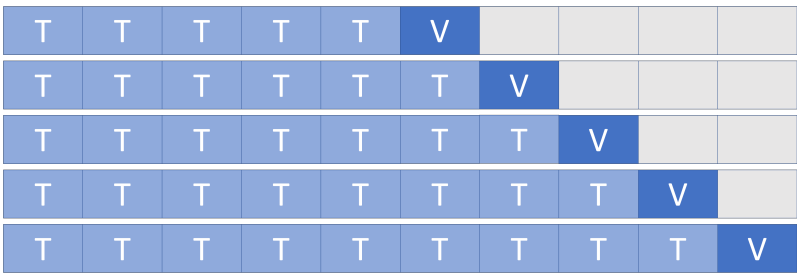
Mean Absolute Error Een mogelijk probleem met MSE is dat ze teveel belang hecht aan outschieters, wat een probleem kan zijn voor generalisatie. Daarom berekenen we ook het gemiddelde van de absolute fout. Deze methode is minder gevoelig voor uitzonderingen en ligt ook in dezelfde schaal van de respons variabele. De MAE wordt berekend als volgt

$$MAE_V = \frac{1}{\#V} \sum_V |f(x) - f^*(x)|$$

MAE wordt minder gebruikt als loss-functie omdat de gradiënt rond nul niet naar nul gaat, wat instabiliteit kan veroorzaken.

Evaluatie Methode

Bij het trainen van predictieve modellen is het altijd belangrijk om het model te evalueren op data waarop het niet getraind is. De simpelste manier om dit te doen is door het model op een vast deel van de data te trainen en te evalueren op de rest van de data. Dit laat echter veel toeval toe in het evaluatie proces. Dit wordt verergerd door het feit dat het gebruikelijk is dat de datapunten willekeurig verdeeld worden over de validatie en train sets. Dit is uiteraard ongewenst bij onze dataset aangezien de punten geordend zijn in de tijd. Dit betekent dat het onrealistisch is om een model te evalueren op data die ouder is dan de data waarop het model getraind is. Een meer diepgaande evaluatie van een model is een kruis validatie: de dataset wordt opsplitst in een aantal 'folds' van dezelfde grootte. Het model wordt telkens getraind op alle folds behalve één, en geëvalueerd op de overblijvende. Maar hier botsen we op hetzelfde probleem als bij een simpele train/validatie split. Daarom kozen we voor een voortschrijdende validatie. Hier selecteren we telkens een periode in de data, trainen we het model op alle voorgaande data, en evalueren we. We zullen dit standaard doen voor de laatste vijf periodes als we de data in tien periodes verdelen. De eerste vijf periodes worden overgeslagen omdat de hoeveelheid trainingsdata, en de gelijkenis met eventueel praktisch gebruik te klein zijn.



Figuur 2: De verschillende scenario's van onze voortschrijdende validatie. Het model wordt getraind met een training set(T) en geëvalueerd met een validatie set(V). De finale resultaten zijn de gemiddelden van de 5 scenario's.

Dataset

De dataset voor dit project is uniek en het samenstellen ervan begon met een lijst van tweets met metadata waarvan de URL de belangrijkste is. Voor al deze tweets hebben we met een scraper gekeken of er een relevante afbeelding bij de tweet gevoegd was, en deze afbeelding gedownload. Aangezien het een grote hoeveelheid tweets betreft en het tijdsgedrag gedomineerd wordt door de vertraging over de internetconnectie, gebruikte dit proces honderden verschillende subprocessen, zodat we de internetconnectie volledige konden satureren. Dit proces resulteerde in een 70000 tal afbeeldingen en tweets die allen het aandeel $AAPL (Apple Inc.) refereerden.

Embedding Met de ruwe dataset zou het heel moeilijk zijn om end to end voorspellingen te doen. Hij is niet zo groot en de kwaliteit van de data is eerder matig, zeker op vlak van de afbeeldingen. Daarom gebruikten we voorgetrainde modellen, voor de twee modaliteiten apart. In multimodaal leren wordt dit beslissingsniveau hybride leren genoemd. We stelden elke tweet voor door een vector van grootte 28, de eerste 24 features kwamen uit de afbeeldingen, de 4 laatste uit de tekst.

Tekst Voor het embedden van de tekst in een tweet kozen we voor VaderSentiment [Hutto and Gilbert, 2014]. Hier werd een lexicon, gecureerd door mensen, gebruikt en werden er verschillende regels en constraints op de tekst toegepast. Dit is een model dat goed presteert op sociale media tekst [Cortis et al., 2017].

Images Voor het embedden van de afbeeldingen kozen we voor een convolutioneel netwerk uit het MVSO project. Deze CNN classificeert afbeeldingen naar Adjective Noun Pairs (ANP's) die we omzetten naar 24 dyades uit Plutchiks emotietheorie. Hiervoor gebruiken we de ontologie van het MVSO project [Jou et al., 2015]. De omzetting naar emoties gebeurde door het gemiddelde van de vijf meest waarschijnlijke ANP's te nemen. We hebben verschillende variaties op deze selectiemethode geprobeerd. Het nemen van de meest waarschijnlijke ANP resulteerde in instabiliteit

bij training en gemiddeldes van meer dan 5 ANP's resulteerde in licht slechtere prestaties.

Tweets	555056
Tweets Multimodaal (voor kuis stap)	71222
Tweets Multimodaal na kuis stap	**56402**
Dagen pre voor opschonen	830
Days na opschonen	**658**
Startdag	2012-12-31
Einddag	2019-09-11
Vorm van data gepresenteerd aan neurale netwerken	$658 \times sample\_size \times 28$
Vorm van data gepresenteerd aan 'naieve' algoritmes	56402×28

Tabel 1: Grootte van onze dataset

Naïeve Toepassing Van Machine Learning Algoritmes

Voor we beginnen met het gebruiken van zeer gespecialiseerde neurale architecturen is het een goed idee om eenvoudig te starten en op een naïeve[1] manier naar het predictie probleem te kijken. We zullen algoritmes gebruiken die geen weet hebben van de temporale structuur van onze dataset. Ze worden tweet per tweet getraind, en ze zullen tweet per tweet voorspellingen maken. Dit zal er wel voor zorgen dat we de dataset presenteren in een andere structuur:

$$D : X \times Y \text{ waar } X : \mathbb{R}^{28},\ Y : \mathbb{R}$$

We werken hier met een aanpak per tweet in plaats van per dag voor verschillende redenen:

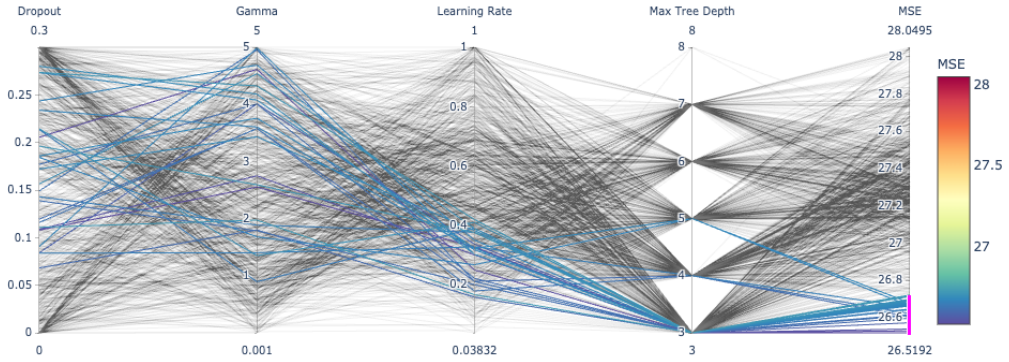
- We hebben maar een beperkt nummer aan datapunten voor onze respons variabele.
- De grootte van tweet sets voor dagen varieert heel sterk, hersamplen met constante granulariteit is hierdoor moeilijker.
- Het laat ons beter toe om deze modellen te evalueren in vergelijking met de gespecialiseerde neurale netwerken.

Lineaire Regressie Één van de simpelste machine learning algoritmes is de lineaire regressie: deze berekent een gewicht voor elke parameter om de kwadratische fout te minimaliseren. Dit algoritme is in tegenstelling tot de meeste machine learning algoritmes niet iteratief en wel deterministisch.

[1] Het is hier de toepassing die naïef is niet de algoritmes

XGBoost Een methode die complexere relaties kan voorstellen is XGBoost: hier gebruiken we Boosting om ensembles van regressiebomen op te stellen. Wij maken gebruik van de Dart booster [Rashmi and Gilad-Bachrach, 2015] die ons toelaat om dropout te gebruiken.

Tuning XGBoost heeft configuratie parameters, ook hyperparameters genoemd, die we moeten optimaliseren. We zullen met een Bayesiaanse optimalisatie de beste configuratie zoeken; het gebruikte criterium is de RMSE van de voortschrijdende validatie zoals eerder gedefinieerd.



Figuur 3: Een diagram dat een selectie van de evaluaties toont, met de 25 best presterende configuraties geselecteerd

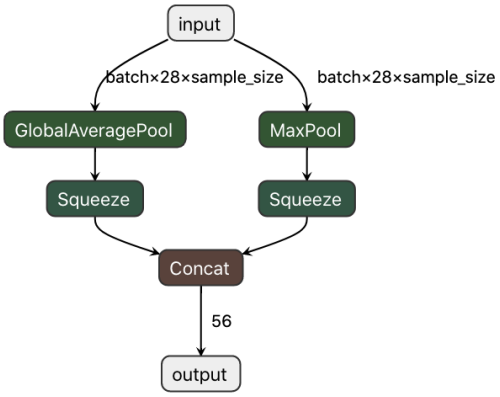
Deep Learning Algoritmes

Na het gebruiken van eerder algemene algoritmes zullen we ook kijken indien het mogelijk is om meer gespecialiseerde architecturen te gebruiken en of die beter zouden presteren. Onze modellen zijn gebaseerd op Convolutionele Middelene Netwerken zoals geïntroduceerd door [Gardner et al., 2019].

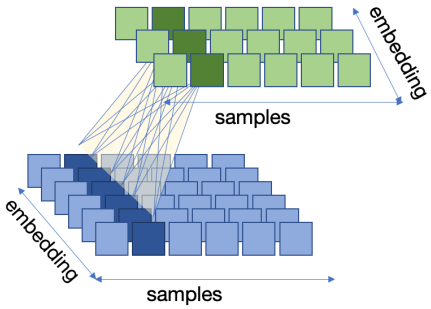
Algemene Concepten

Onze modellen bestaan over het algemeen uit verschillende delen, vooral omdat ze in staat moeten zijn om van een variabele dimensie naar een constante dimensie te gaan.

Embedding De eerste delen van onze modellen zullen de data tweet per tweet beschouwen, ze embedden de tweets in een specifieke ruimte. Hiervoor gebruiken we convolutionele lagen die hier effectief functioneren als een ANN op specifieke tweets. In de meeste netwerken zit het merendeel van de complexiteit in dit deel. Dit zorgt ervoor dat de embeddende functie kan geleerd worden op een aantal datapunten ordes van magnitude groter dan het aantal respons datapunten.



Figuur 5: Implementatie van een aggregatie operatie. Doordat onze input verandert van grootte is de batch size parameter altijd 1. Daarom kan hij weg gesqueezed worden.



Figuur 4: Convoluties op specifieke tweets

Aggregatie Het tweede deel van een model zal een aggregatie uitvoeren op de embedded tweets. Hier reduceren we de onbekende grootte naar een vaste grootte. We gebruiken hiervoor over het algemeen adaptive Max en Avg Pooling.

Regressie Het finale deel van onze netwerken zal de eigenlijke predictie moeten doen, hiervoor gebruikt netwerk meestal één of twee lineaire lagen.

Specifieke Architecturen

We hebben, gebaseerd op de basisconcepten hierboven geïntroduceerd, een aantal modellen ontworpen. Deze worden verder beschreven in de hoofdtekst en appendices maar de bouwstenen worden hier toegelicht.

Mixed Net Dit is een simpele implementatie van de bovenstaande architectuur: het heeft één embeddende laag, één verborgen laag post-aggregatie en een output laag.

Pre Net Dit netwerk doet een voorspelling zonder lineaire lagen na de aggregatie. De finale voorspelling is dus louter het gemiddelde van de voorspellingen voor elke tweet. Het model heeft een residuale embeddende structuur.

Pooling Net Dit netwerk laat interacties toe tussen verschillende tweets voor de aggregatie, het model kan de gemiddelden en maxima van zijn omgeving te bekijken om beter embedden. Het interactie mechanisme wordt getoond in Figure 6

Att Net Veel moderne sequence learning modellen gebruiken attention [Vaswani et al., 2017] met veel succes. In dit netwerk laten we het model toe om te attenden op voorgaande lagen.

Residuale Versies Het residuale Pre Net gaf heel goede resultaten, daarom hebben we andere netwerken met een residuaal embeddend deel uitgerust.

Meer data Alle voorgaande netwerken werkten met enkel multimodale data. Maar we hebben natuurlijk een hoop meer data. We hebben dan ook modellen getraind met meer toegang tot deze extra data: één met alle monomodale (enkel tekst) data en één met de recente historiek van de response variabele.

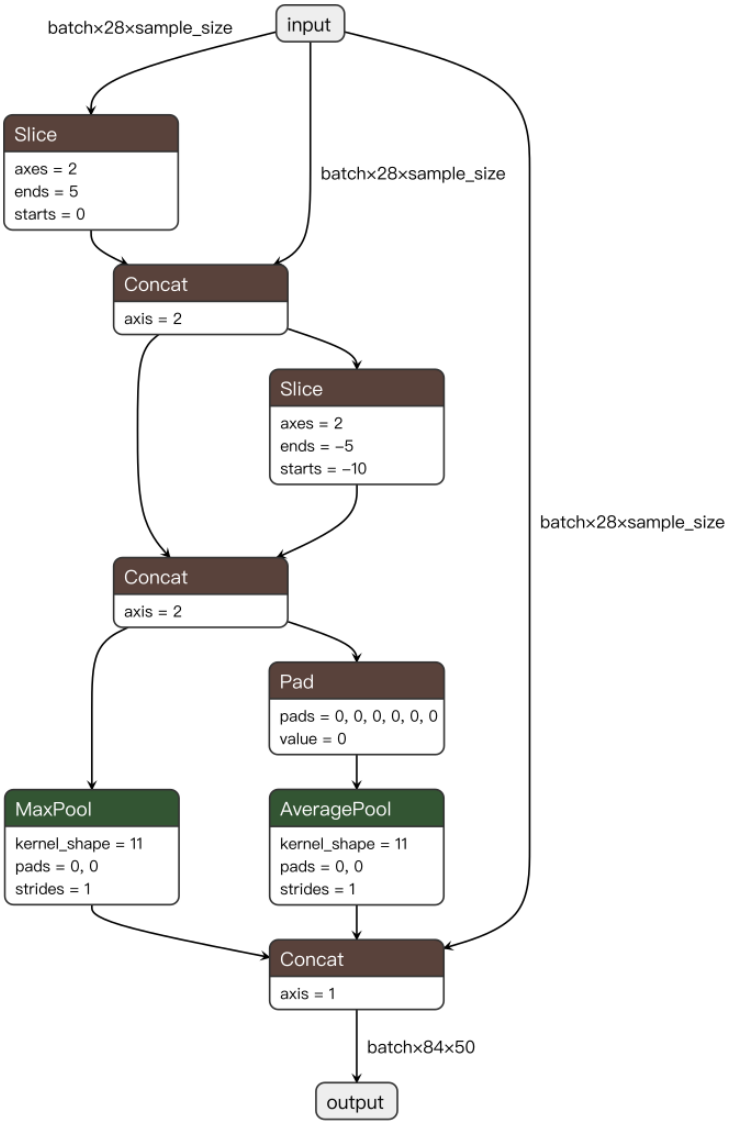
Specifieke Training en Evaluatie omstandigheden

Er zijn veel keuzes bij het trainen van neurale netwerken, en door serieuze computationele vereisten is het niet altijd mogelijk om elke combinatie en configuratie te evalueren. Daarom zijn veel keuzes gemaakt op basis van een combinatie van experimenten en intuïtie. De belangrijkste keuzes waren als volgt:

Loss Functie Voor regressies in gelijkaardige situaties kiest men meestal voor MSE als loss functie, maar we merkten beter gedrag bij het gebruik van Huber loss[2]. Dit is een functie die onder de afstand 1 de kwadratische afstand is, en daarbuiten de lineaire afstand. Dit maakt Huber loss minder gevoelig voor uitschieters, zonder de gradiënt problemen die vaak opsteken bij het gebruik van MAE als loss.

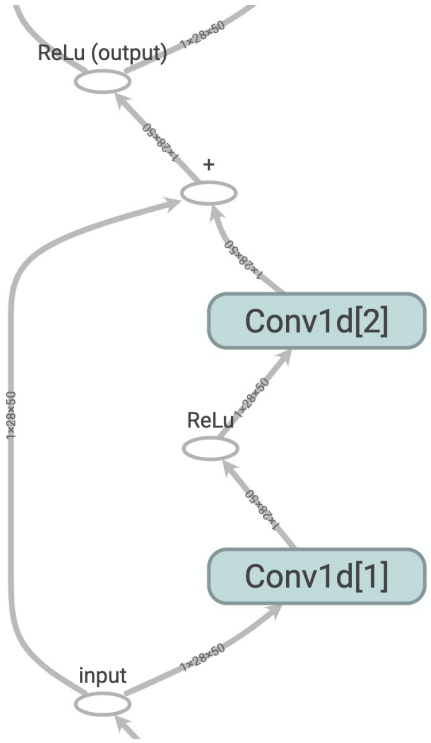
Optimizer Waar de loss functie de fout van de voorspelling bepaalt, is het aan de optimizer om na de achterwaartse pass de gewichten aan te passen. Hiervoor lagen SGD en Adam voor de hand. SGD is de standaard keuze, en bij gebruik van momentum wordt het vaak beschouwd als de optimizer met het beste generalisatiegedrag, zeker voor grote projecten. Adam daarentegen is gekenmerkt door

[2] ook wel smooth L1 loss genoemd



Figuur 6: Diagram van een pooling operatie: we gebruiken circular padding om ervoor te zorgen dat samples aan de uiteindes van het sample op elkaar kunnen inwerken. De input werd geshuffled.

snellere convergentie en er bestaat een aangepaste versie, namelijk AdamW die de generalisatieproblemen tracht op te lossen. In ons project kozen we voor Adam omdat het beter presteerde dan SGD met momentum. AdamW toonde geen significant verschil met Adam.



Figuur 7: Residuale operatie gebruikt in één van onze netwerken. Batch normalisatie werd niet gebruikt aangezien onze batch size 1 was. Experimenten met group normalisatie en layer normalisatie draaiden nergens op uit.

Batch Grootte In deep learning worden inputs vaak in groep (batch) in een model gestoken. De veranderende grootte van onze input zorgt er echter voor dat we een batch grootte van 1 moeten gebruiken. Dit heeft niet alleen negatieve gevolgen voor tijdsgedrag, het zorgt er ook voor dat het model minder stabiel leert. Om dit te vermijden, hebben we gekozen voor een strategie van gradiënt accumulatie, waar we pas een optimizer stap uitvoeren na een aantal inputs.

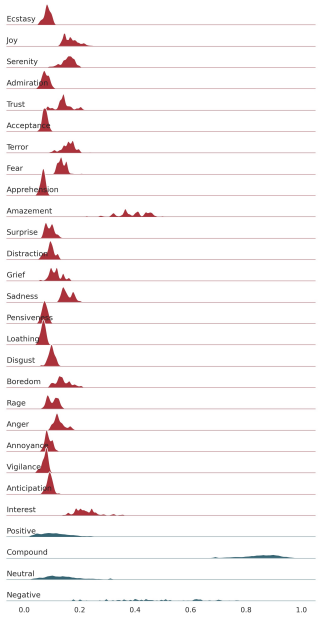
Resultaten

In de vorige delen beschreven we de context, doelen, en opstelling van de experimenten. Hier zullen we kort de resultaten samenvatten, voor meer context en diepgang verwijzen we graag door naar chapter 5.

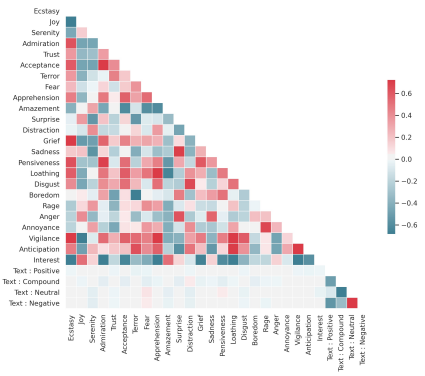
Dataset

In het deel over de dataset gaven we al de grootte van onze dataset mee. Hier tonen we de verdelingen (Figure 8) en interne relaties (Figure 9) van de features, om een indicatie te geven van hun waarde bij voorspellingen. Het feit dat de correlatie tussen visuele en textuele features quasi gelijk is aan 0 is problematisch. Normaal

probeert men de correlatie tussen features zo laag mogelijk te houden, maar we zouden verwachten dat er tussen de textuele en visuele sentimentele waarden een correlatie zou zijn.



Figuur 8: Verdelingen van de sentimentele features, de analyse werd gedaan per tweet na de opschoning



Figuur 9: Correlatie tussen de features, de analyse werd gedaan per tweet na de opschoning

Voorspellingen

We geven in Table 2 de resulaten van onze verschillende aanpakken met enkel de multimodale informatie.

	MSE	MAE	RMSE
Lineaire Regressie	33.607	5.791	4.888
XGBoost	27.625	4.381	5.256
Mixed Net	29.000	4.347	5.141
Res Pre Net	23.119	3.898	4.685
Res Mixed Net	21.260	3.792	4.521
Att Net	26.338	4.183	4.980
Pooling Net	21.219	3.633	4.496
Pooling Pre Res Net 5	22.085	3.718	4.591
Pooling Res Net 5	**19.4516**	**3.473**	**4.157**
Pooling Res Net 20	23.445	3.837	4.847
Pooling Net Plus*	19.692	3.567	4.302

Tabel 2: Out of sample resultaten van alle modellen.
*dit model gebruikt meer (monomodale) informatie

Realistische Prestaties Vergeleken met de interne voorspellingskracht van de response variabele zelf zijn onze modellen niet heel performant maar dit is te verwachten aangezien we een redelijk kleine, ruwe dataset gebruiken. Als we de IV van de voorgaande dagen aan het model geven, zien we dat het beter presteert dan gewoon de waarde van de dag ervoor nemen als voorspelling.

	MSE	MAE	RMSE
Pooling Net Res 5 met voorgaande IV's	8.522	2.172	1.633
Pooling Net Res 5 zonder voorgaande IV's	19.4516	3.473	4.157
Presented Dataset	13.720	2.350	3.704

Tabel 3: Resultaten als men op een (licht) realistischere manier werkt.

Overige Resultaten We verwijzen graag naar het hoofdstuk resultaten in de hoofdtekst voor resultaten van verdere experimenten. Hier illustreren we sommige keuzes die we gemaakt hebben, eigenschappen en structuur van onze dataset, en het specifieke tijdsgedrag van de evaluaties.

Conclusie

Dataset

Bij de constructie van de dataset en de extractie van de sentiment scores zijn er weinig compromissen gemaakt. Het was een mogelijkheid geweest om de dataset zoveel mogelijk op te schonen maar dit zou de resultaten minder realistisch maken. De methode die we gebruiken voor de textuele analyse, waar men normaal veel

preprocessing en cleaning nodig heeft, heeft dit ook gewoon minder nodi.g De finale dataset is uiteindelijk niet zo groot en redelijk ruw, veel afbeeldingen dragen slechts weinig betekenis en sommige helemaal niet. Het taalgebruik van financiële Twitter accounts is ook redelijk gespecialiseerd, alhoewel Vader uiteindelijk goed presteerde. Een grotere dataset is duidelijk een mogelijkheid voor verbetering, en we zijn van het oordeel dat de methodes en processen uit dit onderzoek hier zeker voor gebruikt kunnen worden.

Machine Learning

Het is duidelijk dat onze gespecialiseerde modellen algemene algoritmen overtreffen, ze presteren echter niet goed genoeg om echt van praktisch nut te zijn. Dit is echter niet onverwacht: modellen in productie met dezelfde taak zouden nooit enkel tweets beschouwen. Ook demonstreren we dat als historische informatie aan ons model toegevoegd wordt, het de variabele waaraan het werd blootgesteld overtreft.

Finale Gedachten

Financiële variabelen voorspellen is moeilijk. Maar ondanks een kleine en ruwe dataset concluderen wij dat multimodaal sentiment, en de modellen voorgesteld in deze tekst, duidelijk waarde hebben voor het voorspellen van financiële variabelen, met name de geïmpliceerde volatiliteit.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations

Abbreviations

ANP	Adjective Noun Pair
CDAN	Convolutional Deep Averaging Network
CNN	Convolutional Neural Network
EOD	End Of Day
ETF	Exchange Traded Fund
IV	Implied Volatility
MAE	Mean Absolute Error
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
VIX	CBOE Volatility Index on S&P500 index [Exchange, 2009]
VXAAPL	CBOE Volatility Index on $AAPL
XGB	XGBoost [Chen and Guestrin, 2016]

Chapter 1

Introduction

Accurately predicting future situations is a key application of learning algorithms, and financial markets are some of the hardest areas to make those predictions in. This is not just caused by their sheer complexity, tight coupling with the global socio-economic system, and the sheer amount of hidden information and mechanisms. Accurate predictions of future financial situations with publicly available information imply a better understanding of the market situation than the aggregate market itself.

1.1 Oracles, and their Demise

For a long time active fund managers were assumed capable of consistently outperforming the market, which allowed them to charge large fees. These often consist of management fees and fees on excess performance compared to benchmarks. Their customers assumed that through experience and deep understanding of the financial system and world they would be able to outperform less sophisticated investors. This idea is however being challenged, as more and more investors are choosing for passive funds, managed by simple constraints and rules. They could for example follow an index, such as the S&P500 being followed by the $SPY SPDR S&P 500 ETF. Their passive management allows them to charge lower fees to their investors. It has been demonstrated that historically the majority of actively managed funds fail to consistently outperform passive funds [Malkiel, 2003b]. This can be characterized as a failure to maintain a positive alpha by the fund manager. Alpha is a measure of the performance of a portfolio that also considers the risk taken. Positive alpha means that the portfolio provides returns higher than a comparable index, without more risk being assumed, and thus indicates a manager consistently able to make the right calls, and provide returns beating the market without just making riskier investments. When looking at actively managed funds, it becomes clear that only a small amount of elite managers can provide these consistent excess returns to justify their fees. Another class of actively managed funds are hedge funds, these often take very specific positions trying to provide their investors with some alternative investments that could stabilize their portfolio, but their performance at this task is

not uncontested.

The past decade has shown an increased appetite of investors for ETF's with their lower fees and complexity. The difficulty of consistent market-beating returns is often linked to the efficient market hypothesis. It posits that the market, in commonly traded sectors, is efficient at determining the value of products. This means that it becomes difficult to predict price changes when only considering information that is available and used by the majority of actors in the market, and thus the only way to generate higher returns is to either know more or assume more risk. This difficulty is also seen in naive projects using deep learning algorithms to predict the stock market. They often fail to capture any complex patterns and often struggle to outperform simple machine learning algorithms.

1.2 Alternative Data for Alternative Insights

Recently a new approach to know more than competitors[1] has been gaining traction. Enabled by the internet, more sensory data and cheaper computational resources, analysts have started gathering alternative data, information not commonly considered by the market, to make better decisions. It often consists of large amounts of data somewhat indicative of some aspects of a security or the underlying company. It can be contrasted to privileged information, which is internal information of a company not (legally) considered by the market.

Alternative data is commonly not explicitly released by the company and can be used to predict or estimate some aspects better than the market can. These datasets are usually very large and often require significant processing to extract information. They could be by-products of the functioning of a business, such as shopping receipts, credit card transactions or website visits. Other possibilities to construct them include satellite imagery or web scraping.

Something relevant for picking stocks or compiling portfolios is the sentiment of the market toward certain products. While predictions of analysts at big financial institutions are often priced in, the aggregate sentiment of investors is less easily measured, although it can have a big effect. A type of alternative data that can be used to estimate the sentiment of the market for specific securities is data published on social networks. Of this kind of data the prime example are 'tweets' published by users on Twitter, a very popular microblog. Sentiment analysis based purely on text is common these days, but image-based sentiment is not[2]. The combination of these data modalities is also not yet commonly applied in the area of financial predictions. In this text we will analyse different modalities of data and try to use them to better estimate sentiment and predict volatility.

[1] Or to be able to compete with competitors already applying these strategies

[2] Other than inferring sentiment from facial expressions

1.3 Course of Action

In this project we will compile a dataset of tweet-image pairs referencing an equity ($AAPL), we'll extract sentiment from the multiple modalities present in the dataset. We will evaluate the usefulness of this dataset to predict implied volatility on that equity for the day following the publishing of the tweets. The performance of convolutional deep averaging networks (CDAN's) with modifications will be evaluated versus simpler machine learning approaches in the prediction of this implied volatility. The scope of this project will be constrained to using only the multimodal data, for predicting the implied volatility of the next day. We will try to evaluate the potential use of both multimodal sentiment and presented models in a practical setting, where predictions are performed every day on new information.

1.4 Structure

chapter 2 will introduce concepts vital to this project, after which chapter 3 will list relevant works not yet covered both on predicting Implied volatility and on multimodal sentiment. chapter 4 will outline the approach of this project, starting by defining the goals of this project and the metrics by which we will evaluate our result, followed by the approaches and algorithms central to this project. It continues with the construction of our dataset, and the approach to extracting sentiment features. It then describes the design of the experiments with predictive models: First, as baseline, a traditional approach is used, followed by specialized deep learning networks. The results of these experiments are presented in the penultimate chapter. Finally, the final chapter describes insights gained, puts results into context and lays out possible areas of future research.

Chapter 2

Concepts

2.1 Efficient Market Hypothesis

When looking at mature financial markets it's often said they are efficient, meaning that products somehow get the correct price given the available information, it is however not easy to define the correct price. A weaker statement is that the market can be considered as a functional relation, meaning that it results in a price for every set of considered information. This considered information consists not only of the fundamental information about a company or its sector but also the state of the market, and general sentiment.
The system of market prices is kept largely consistent by actors trying to outcompete each other to profit from the inconsistencies. It is however important to realize that they are mostly following monetary incentives which can cause market prices to also depend on market conditions. A good example of this is the negative price of crude oil futures with delivery in May 2020 days before they expired in April. The negative price here was due to the fact the holder of these futures after the expiry is forced to receive a large amount of physical oil. A lot of these futures are however bought by investors only speculating on the oil price. The preceding crash in demand and price had caused storage to be filled, meaning storage capacity was so limited it was valued higher than the oil it contained was worth.

2.1.1 Consequences for Machine Learning

The prediction of financial variables is something a lot of very smart people have spent a lot of time on. If one could predict a significant change in for example price with a arbitrarily large amount of certainty, he could exploit this change with an arbitrarily highly levered position. A perfect prediction would thus imply arbitrarily high profits. Alas, the world is not this simple, these kind op arbitraging possibilities are rare, and taking a position will influence the market which will prevent us from making infinite profits. The reason that these opportunities are rare is expressed by the previously expressed efficient market theorem. It is these effects that doom the naive application of even very complex machine learning algorithms to trading. They can simply not compete with the price-setting power of the market.

2.2 Implied Volatility

Financial product prices always have an inherent variability, a simple measure of this is volatility or standard deviation. This is a backwards-looking measure, indicating the amount of movement in prices over a certain time frame. When trading, estimating volatility in the time to come is very important. One place where this is the case is in option pricing. Options are contracts on some underlying product allowing the holder to buy or sell an amount of that product at or before a certain time. Mathematical methods for estimating the price of a certain option generally require at least an estimate of the volatility in the time frame of the option. This means that the price of an option implies some volatility in the future. This implied volatility is an indicator of the volatility expected and priced in by market actors. When we want an aggregate view of volatility expectations we can solve an option pricing problem in reverse, inputting the prices and solving for the volatility. The actual calculation used for the values will be described in section 4.2. One example of implied volatility is the VIX index calculated by CBOE, reflecting the volatility expected by the market based on 30-day options on the S&P500. This index is sometimes called the fear index, as a high value indicates uncertainty in the market. It is however important to understand that it does not imply a direction for the expected movements.

2.2.1 Usefulness of Implied Volatility Predictions

In practice implied volatility predictions would likely be used as a part in a more complex trading system, it is however possible to trade on implied volatility alone. Option prices, as previously mentioned, depend on implied volatility, and changing implied volatility will often cause changing option prices. The derivative of an option's price with respect to the implied volatility of the underlying asset is called vega: $\mathcal{V} = \frac{\partial V}{\partial \sigma}$. It's calculated for a European option following the Black-Scholes-Merton model as follows: [Hull, 2018]

$$S\phi\left(d_1\right)\sqrt{T}$$

Where:

S	Stock Price
T	Time to expiration (years)
r	Risk-free rate
K	Strike price
σ	Volatility
$\phi(x)$	$\frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$
d_1	$\frac{\ln(S/K)+\left(r+\sigma^2/2\right)T}{\sigma\sqrt{T}}$

One could use this vega to estimate the behaviour of positions when considering the predicted IV change. Implied volatility could be speculated on by using options

structures such as straddles or strangles. If one wants to concentrate exposure to implied volatility however another possibility is to speculate on (conditional) variance and volatility swaps. The usage of these approaches and instruments is however subject to a lot more consideration. They are mentioned here solely as simplified examples of exploitation of sufficiently capable implied volatility.

2.2.2 Multimodal Learning

Multimodal learning is a relatively recent area of research where the goal is to effectively learn across multiple modalities of information. This is trivial for humans, as they efficiently combine information from sensory receptors detecting stimuli in vastly different modalities. These stimuli vary from light (visual), pressure waves (sound) to chemicals (smell) and even inertial effects (acceleration and orientation). Designing algorithms to perform this sensory fusion is however less trivial, approaches to these problems, specifically in the area of sentiment extraction will be documented in chapter 3.

2.3 Sentiment Extraction

Humans are emotional creatures, and most of their communication carries emotions or sentiment. While it's often trivial for a human to evaluate the sentiment of communication this becomes a lot harder when it has to done by algorithms. Sentiment rests on a set of shared understandings and knowledge, and understanding it requires deep knowledge of the human experience. It is very attractive to have algorithms that understand the emotional load of communication. In this project we focus on two different modalities of communication, textual and visual, and try to use them to make IV predictions. There are different ways of representing sentiment, the simplest one would be a simple polarity, expressing the positivity or negativity of a sample. But just because a message is not positive it doesn't mean it is negative it could also be neutral or non-emotional. This implies that there are multiple values we could use to represent sentiment polarity. Sentiment polarity is not the only emotional information in a given message, humans experience a lot of emotions that vary in more than the basic polarity spectrum. A common model is Plutchik's wheel of emotions, it uses 8 primary emotions and models a total of 24 'dyads' or composite emotions[Plutchik, 1991]. Sentiment is often embedded in a vector of these 24 dyads, with each number representing the strength of that emotion. We will detail sentiment extraction approaches specific to different modalities in chapter 3

2.4 Machine Learning Algorithms

2.4.1 Linear Regression

Linear Regression likely is the simplest machine learning algorithm, Also called parametric regression it learns a weight for every input variable to minimize squared error to the response variable.

2.4.2 XGBoost

A very simple, very common prediction model in machine learning is a decision tree, they are often used for predictions, are very interpretable, light weight and easy to understand as they closely mirror human decision-making. They however struggle to generalize complex concepts, and are not very robust to variations in the dataset. A common approach for improving the performance of weak learners is to construct ensembles of them to try to improve some or all training characteristics. These ensembles can be constructed in many ways but a very popular one is to use boosting. Here we work incrementally and try to correct the mistakes made by a model. This allows us to use weak learners, whose response is only slightly correlated with the actual response variable, to construct a strong learner, where the response and the true response can be arbitrarily well correlated. The specific (meta) algorithm used is gradient tree boosting [Breiman, 1997, Friedman, 2001]. Here we will try to minimize a loss function (MSE) by performing gradient descent. The implementation used is XGBoost [Chen and Guestrin, 2016], which is very popular for similar, feature-engineered applications.

Hyperparameters

Hyperparameters are parameters of an algorithm that are set before the learning process, and can not be inferred from the training task, they generally influence the training performance. Gradient based tree boosting has a lot of these variables, and we chose some to focus on.

Tree depth The max depth of a tree in the ensemble, increasing this increases complexity expressible but makes overfitting more likely, the complexity also increases with $O(2^{Tree\,depth})$.

Eta Also called learning rate, controls weight shrinking between training steps. Higher values make the algorithm more conservative.

Gamma Controls the growing of trees, higher values make the model more conservative, this value is often used to try to control overfitting while using high tree depth values.

Dropout rate The fraction of trees that will be dropped after an iteration. This is a value specific to the dart booster [Rashmi and Gilad-Bachrach, 2015] and is inspired by dropout in deep learning.

2.4.3 Validation

The simplest way of checking the performance of the model on unseen data is to keep a validation part, train the model on the rest of the data, and then assess the performance on the validation part. This however can be generalized in a cross

validation. Here we divide the data into an n folds, and we train the model one time for each fold, each time with one fold as validation set and the other ones as training set. This process is called n-fold cross validation, and is very common in the assessing of performance of learning algorithms.
Our data however has an inherent temporal order, which means that this kind of testing is not representative of how the model could be used in the real world. To account for this we will use a walk forward validation. Here we will again divide the data into a number of testing folds but now explicitly without shuffling, which is commonly done in normal validation. When training a model, we will also only train the model on the data preceding the validation fold. We have done this in 2 different ways, one way were we divided all the data into folds and one where we selected a part of the data that was always part of the training set. This was done because the average loss seemed to be dominated by earlier folds with a low amount of training data.

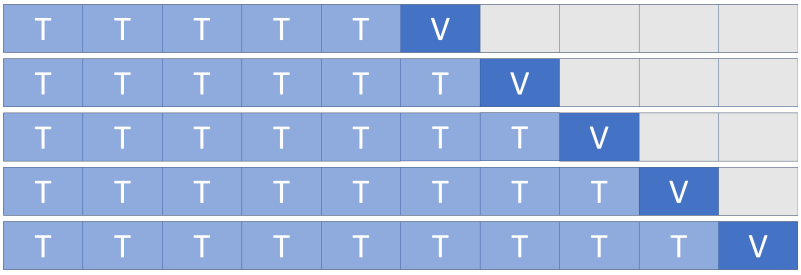


FIGURE 21: Our validation will consist of 5 training scenario's each with a training(T) and validation(V) set. The final result will be the mean of the scores.

2.4.4 Bayesian Optimization

One thing necessary to properly asses how powerful models with hyperparameters are is a hyperparameter optimization. Traditionally this was done naively with grid or random searches. The curse of dimensionality means however that this will quickly result in a lot of compute time necessary for even a fairly rough estimate.
This is why we use a method that is able to leverage the previous evaluations to choose the most interesting places to evaluate. We used Bayesian optimization, with this method we try to model the loss surface and sample only points that are likely to give us a lot of information about the loss surface or could be a possible optimum. To determine the loss for a hyperparameter selection we used the same walk forward validation as in the evaluation stage.

2.5 Deep Learning

Deep Learning is a subset of Artificial Intelligence seeing a lot of attention in recent decades. The original base of deep learning was the Artificial Neural Network (ANN),

a network organized in layers of Perceptrons, which model biological neurons. Basic neural networks have a number of layers, each with a number of nodes, and every node gets all the outputs of the previous layer, multiplies it by a weight vector and applies an activation on the result, this is its output, which is passed on to the next layer. These networks are called fully connected feedforward networks, and deep learning was originally understood to concern the research of neural networks with more than one hidden layer, with a hidden layer defined as a layer that is not the input our output layer.

Universal Approximation It can be proven that ANN's can approximate any function given some constraints, as stated by the Universal Approximation Theorem: which says that ANN's with one hidden layer and some mild assumptions on the activation function used are a universal approximator for continuous functions on compact subsets of $\mathbb{R}^n$. It can be stated that if our approximator can be described as

$$f^*(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

with $\varphi : \mathbb{R} \times \mathbb{R}$ a valid activation function.[1]
Then for every $\varepsilon > 0$ and every real valued, continuous function on a compact subset of $\mathbb{R}^m$: f there exist a width $N \in \mathbb{N}$, a weight vector $v \in \mathbb{R}^N$ and a bias vector $b \in \mathbb{R}^N$, so that

$$|F(x) - f(x)| < \varepsilon$$

2.5.1 Layers

Neural networks are often considered to be organized in layers, in early networks these layers had clear order, and every layer was connected with the layer before and after it. Modern networks however have more complex graphs, with increased connectivity, skip connections and multiple paths being common. Some examples of networks that use advanced connection schemes are ResNets and the networks powering the deep learning part of the tesla autopilot [Karpathy, 2020]. While these days these layers are more node than layer the nomenclature stuck around, we will now introduce some relevant layers used in our experiments.

Linear Layer the base layer, also called a dense layer or a fully connected layer. This is the simplest layer which performs a multiplication of its input with a weight matrix and adds a bias vector. This layer, while being one of the simplest ones is also one of the most expressive layers used. It denotes an operation:

$$L(x) = xW^T + b \text{ where } x \in \mathbb{R}^{d_{in}}, W \in \mathbb{R}^{d_{out} \times d_{in}}, b \in \mathbb{R}^{d_{out}}$$

This means that it has $O(d_{in} * d_{out})$ trainable weights, which can become expensive for large dimensions. After training a very big network, it is often possible to prune away a lot of the weights with only minor changes in performance.

[1] This is commonly the Rectified Linear Unit (ReLU) but can be all non polynomial functions.

Convolutional Layer Linear layers have big difficulties when applied to data with strong spatiality, meaning that for representing features the nearby features are the most relevant. This is the case when for example images, or sound is used as input. Convolutional layers are sometimes intuitively explained as applying filters to the input but it can be more rigorously described as applying connectivity restrictions and weight sharing to a linear layer. In a linear layer all input variables are connected to all others. This means that a certain output variable is a function of the whole input, we can say that it's receptive field is the whole input which in images with strong spatiality is largely non-relevant. This means that we can limit the receptive field. The size of this receptive field is called the kernel size and has the same number of dimensions as the spatiality of our data. Another important insight is that the features in one spot will be very similar to the features in another spot, to exploit this we can share weights, essentially learning the same features in different locations. Both of these changes will, in fitting modalities, seriously improve our learning ability while seriously decreasing computational complexity. These convolutional layers will combine several kernels, meaning multiple of the constrained linear layers applied, and are the foundation of modern computer vision [Fukushima, 1988] [LeCun et al., 1989].

Pooling Layer these layers are very common in convolutional neural networks. Convolutional layers often are not configured to perform serious dimension reduction, so this is done by the pooling layers between (groups of) convolutional layers. A pooling operation consists of the applying of a pooling function $p : \mathbb{R}^{mn} -> \mathbb{R}^{m}$ over the input, the most common functions used are the mean and max functions. Two common ways exist of specifying this operation, a fractional one where one specifies the kernel of the operation which will determine the dimension reduction, and an adaptive one where one specifies the output size, letting the deep learning framework figure out the kernel size during inference.

Distributive Pooling Layer This is a variation of the pooling layer we implemented and used in some of our models is distributive pooling, here we pool the input to a dimension of another tensor, while this seems trivial to implement by just using an adaptive pool to the relevant size, the implementation becomes a bit more complicated. We will have to perform control flow based on an input which is something PyTorch is able to do[2] but requires some additional attention. To assure correct behaviour we need to wrap it in a TorchScript module. This, among other things, tells PyTorch that all control flow in that module should be considered part of the graph, as otherwise control flow is considered to be constant between forward passes.

Residual Blocks Neural networks often perform better when their depth is increased, but the learning performance in very deep networks is often diminished, due

[2]This is because of its dynamic computational graph, which is one of its standout features, and is being implemented in some other frameworks (e.g. Tensorflow 2) that previously had static graphs

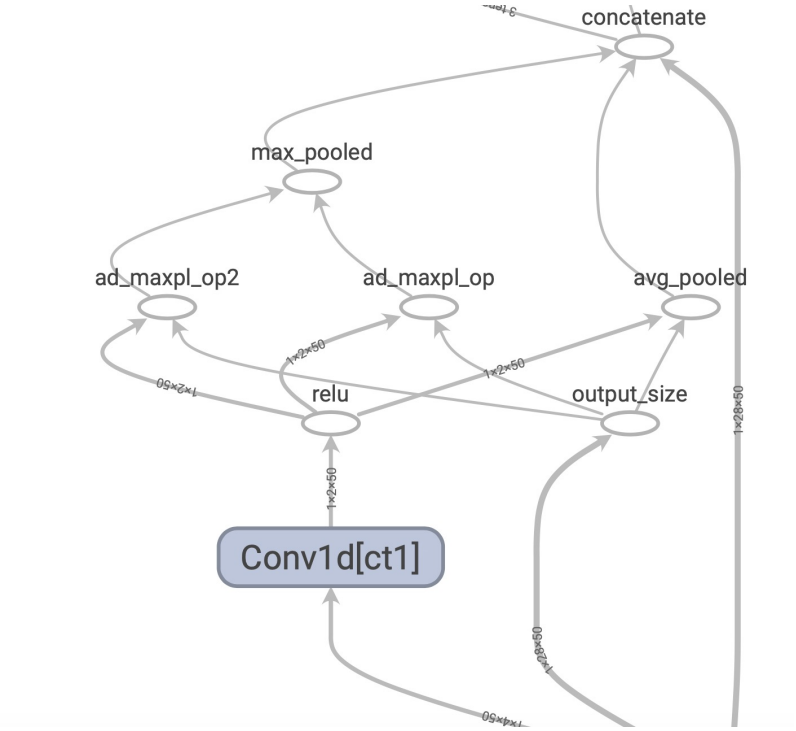


Figure 22: Computational graph describing a possible way PyTorch could construct our distributive pooling operation. Data is effectively distributed over different multimodal samples, after which they are pooled.

to the strength of the gradient being diminished, a commonly used solution to this is to introduce connections that leverage skip connections [He et al., 2016]. Networks that feature these skip connections are better known as residual networks, and their layers are generally organized in blocks.

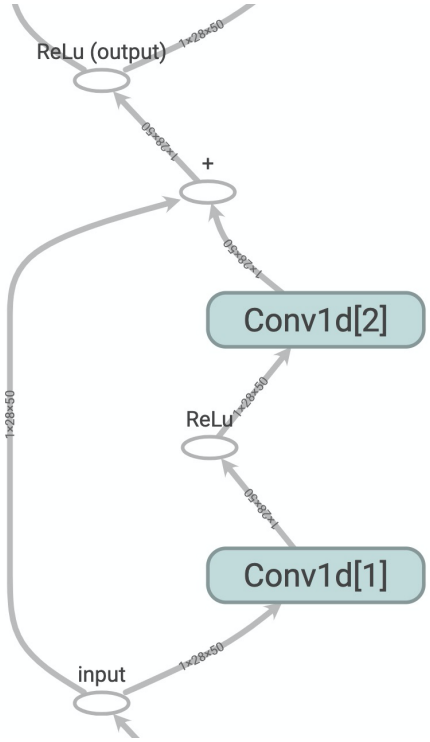


FIGURE 23: Computational graph for a residual block used in one of our networks

Chapter 3

Related Literature

3.1 Implied Volatility Prediction

As previously mentioned predicting financial parameters is hard, and accurately predicting changes in implied volatility implies that one has a better understanding of the market than institutions pricing the options itself. There is not a lot of published research on this specific problem, there is however research on the usage of machine learning to leverage tweets in financial forecasting of other variables. [Audrino et al., 2020] demonstrates the use of microblog sentiment in addition to attention measures (e.g. google trends data) for forecasting volatility, [Sardelich and Manandhar, 2018] proposes the usage RNN's and different modalities of information, [Oliveira et al., 2017] shows that commonly used sentiment surveys can be augmented with microblog sentiment using a Kalman filter. While some researchers [Behrendt and Schmidt, 2018] conclude the benefits of alternative textual information to be economically negligible, the consensus view is that there is a strong correlation between sentiment and variables like volatility, and that including microblog sentiment estimates can result in a small but significant improvement in short term forecasting performance. Benefits are mainly seen with companies that have a lot of non-institutional investors, with which the public has a significant sentimental connection.

3.2 Sentiment Analysis

Multimodal sentiment extraction is a field seeing a lot of attention in recent times, and much of the work focuses on audiovisual sentiment [Poria et al., 2017a], while work in unimodal settings has traditionally focused on the textual modality.

3.2.1 Textual Sentiment

From a practical standpoint text often conveys the clearest sentiment, and over history there have been a lot of approaches, which can be divided into two categories.

Rule-Based

The original way to perform sentiment extraction was, as was most of early NLP, based on constraints and features built by humans. The algorithms relied on knowledge deliberately compiled by humans. Simple approaches to this are so-called Bag Of Words models (BoW), these represent all words in a text separately and do not consider interaction between words. These approaches underperform humans but not by as much as could be expected [Mishne et al., 2005]. The logical next step is to leverage the syntactical structure of language and model more complex features. This could mean using resources to parse text to provide Word Sense Disambiguation, or model things like negations [Jia et al., 2009] or amplifications [Rentoumi et al., 2009] [Hung and Chen, 2016]. One example of a rule-based approach is Vader [Hutto and Gilbert, 2014] which combines simple syntactical structure and a human-curated lexicon with a couple of simple rules about human speech, this approach is simple but performant, especially in social media applications. The quality of the lexicon is demonstrated by it being used by top-performing teams [Mansar et al., 2017] [Cabanski et al., 2017] in the 2017 SemEval competition task [Cortis et al., 2017] for the analysis of the sentiment in financial microblogs and headlines [1].

Machine Learning Based

Recent research in NLP has focused on deep learning techniques, and powered by the GPU revolution, NLP models have grown exponential and have come to rival humans in performance. Transformer based models are currently the state of the art for a lot of NLP tasks [Devlin et al., 2018] [Radford et al., 2019]. These models generally use self attention and are often trained to predict masked parts either in or at the end of sequences. And while not originally trained for sentiment analysis they can be fine-tuned on the task [Howard and Ruder, 2018], these fine-tuned transformer-based language models form the state of the art on general sentiment analysis tasks. [Raffel et al., 2019] The representation of text going into these models is often character or word level, a subword representation using Byte-Pair-Encoding can also be used.

3.2.2 Visual Sentiment

Visual sentiment analysis is currently not as big a field as its NLP counterpart. But some of the same principles apply, most approaches here are deep learning-based and often rely on similar principles as image classification or regression techniques. Models can try to predict straight to emotional values (e.g. polarity, or specific emotions) but often features are classified into an ontology modelling entities, concepts, and aspects from which sentimental features can then be extracted [Chen et al., 2014a] [Campos et al., 2017] [Chen et al., 2014b]. An ontology commonly used is MVSO

[1] While subsequent SemEval competitions have taken place the specific task didn't feature in them

[Jou et al., 2015] compiled by researches at the University of Columbia, strongly related to the Sentibank [Borth et al., 2013] project, whose dataset is a common benchmark in visual sentiment analysis. The pretrained models and datasets from these projects are commonly used in models with visual modalities [Kumar and Garg, 2019] [Wang et al., 2016]. MVSO contains pretrained models that classify images into a number of concepts, called Adjective Noun Pairs, and an ontology that allows us to convert these into an emotional vector with 24 values, representing the 24 dyads from Plutchiks wheel of emotion[Plutchik, 1991]. The most common area of research in visual sentiment analysis, while not being relevant to this work, is facial analysis [Poria et al., 2017a].

3.2.3 Multimodal Sentiment

Multimodal learning and specifically multimodal sentiment are relatively recent areas of research within artificial intelligence, where the goal is to learn representations across multiple modalities of information. Until recently most AI research effort was put into methods with information of only one modality. Most research focuses on combinations of visual (both images and videos), auditory, and textual information. For multimodal learning, and more specifically multimodal sentiment predictions there are two main approaches, with hybrid approaches using both concepts.

Decision Level Fusion means performing the sentiment prediction for both modalities split, after which the sentiment is combined. If a human would for example try to use this approach to detect a traffic incident in a video he would first watch the visual part, looking for a traffic incident. Then he would write down his conclusion, forget all about the video and listen to the audio listening if he can hear an accident. He would then finally combine his findings to make his decision of whether an accident occurred or not. While this means we lose context within our specific predictions it also allows us to use more specific processes, that do not need to be compatible with the other modalities. [Cai and Xia, 2015], representative of the common approach, proposes conventional classification/regression models on features from split models, while [Glodek et al., 2013] leverages a Kalman filter.

Feature Level Fusion will combine the modalities at the start of the process, this means the different modalities can fully interact with each other. While this obviously could allow better learning, it also requires a model to be able to handle the combined features of both modalities. This often means that pretrained monomodal methods, which are often more advanced than dedicated multimodal ones, cannot be used. A common way of achieving this, demonstrated by [Poria et al., 2015] is concatenating the features. A person obviously could not try to use this method to detect accidents as in our previous example as he will have a lot of modality specific processing before the information is fused. It is this that pushes us to use a hybrid approach, between the two extremes.

Model Level Fusion is a combination of the previously described models, and handles the sensory fusion inside its model, using intermediate, modality specific, processing to represent the information in a way it can be fused, but also interact with the other modality during its processing. This is of course optimal, but it also requires the most complicated structure. This would also be how a person would detect an accident, combining visual and auditive stimuli to come to a decision. Internally these models will use both decision level and feature level fusions, as demonstrated in [Poria et al., 2017b].

3.3 Data Challenges

3.3.1 Constraints on Financial Data

One problem that arises a lot in financial time series is the general lack of datapoints. A lot of approaches use an end of day (EOD) granularity. Since a lot of data is only available from the 2000s onwards this means that time series have relatively small sizes.[2] In the case of alternative data the datasets often only look back a decade, only aggravating this problem.

3.3.2 Set Learning

When working with multiple samples for every datapoint a common approach is to aggregate the data into a single vector, for example averaging it. This aggregation allows us to use traditional models with constant input sizes and counteracts the noise on single samples, it's however possible that a lot of information is lost in the aggregation. The opposite approach has a lot of problems of its own, working with single samples makes the process vulnerable to noise, irrelevant samples, and, if the number of samples within datapoints varies a lot, a bias towards response values that have a lot of samples.

Multiple Instance Learning

Faced with the problem of a limited number of datapoints and difficulties with both learning at EOD granularity or at tweet granularity one can wonder if there are no approaches that can consume all data with EOD granularity, while still leveraging all the information given by the varying number of samples in the datapoint. This is a problem that not only exists in our specific problem, and different architectures have been proposed to elegantly overcome these challenges.

Sequence Learning One option is just to forget about the fact that our input is unordered and use a sequence model, here RNNs are the most common, but attention-based models are gaining traction. The main issue with this approach (more with the RNN but also with attention-based models) is that models learn patterns present that are non-meaningful, RNN's also are generally tough to train on

[2] Between 01/01/2000 and 01/01/2020 there are only about 5060 trading days.

longer sequences [Hochreiter et al., 2001]. The consensus is therefore that specific models aimed at unordered datasets are needed.

Unordered Set Learning Architectures specifically designed for unordered sets are not as well studied. One important property that is desirable here is permutation invariance, meaning the model's output does not change when permuting (shuffling) the input. An early example of a permutation invariant approach are BOW algorithms, used in NLP, and this approach has formed the inspiration for more complex models such as modified recurrent networks [Richard and Gall, 2017] DANs [Iyyer et al., 2015] and CDAN's [Gardner et al., 2019]. The CDAN, in its basic form, performs independent embeddings of samples after which a pooling operation aggregates the samples and a normal neural network performs the actual prediction. This project also proposes architectures that sacrifice the permutation invariance and independent embedding to allow for interaction between samples. Related architectures also feature in the application of deep learning on point clouds with PointNet [Qi et al., 2017] and dataset representations with a variational autoencoder [Edwards and Storkey, 2016].

Chapter 4

Approach

4.1 Problem Statement

Previously we stated that we aim to predict financial variables, specifically implied volatility, from sentiment extracted from multiple modalities in tweets. This chapter will specify the problem to be solved in this thesis and describe our goals in this problem.

4.1.1 Extracting Features

Our first goal is to actually create a dataset for this task, and then embed it in some meaningful feature space enabling us to train models and study the relation between implied volatility and our measures of sentiment.

Goal 1. *Construct a process that, when given a set of tweets, can construct a multimodal dataset and extract sentimental features usable for an implied volatility prediction.*

Practically this means we will construct a collection of tweets, organised by day. We will extract sentiment in both the visual and textual modality, and we will represent this in a vector of constant size where each value gives us some emotional information about the tweet it was based on. We will evaluate if and how we need to preprocess and clean the data before sentiment extraction, and how we can configure and tweak the representation construction for the dataset to be as useful as possible.

4.1.2 Predicting Implied Volatility

After getting the actual dataset we want to use it to see if we can predict the future implied volatility for financial products.

Goal 2. *Given one or more tweets with text and images, can our model predict the implied volatility for the next business day?*

Practically this means we will experiment with algorithms to predict the Implied Volatility on a specific equity ($AAPL) at market close using only the sentiment extracted from tweets posted the previous day.

Data After feature extraction our data will be used as a multidimensional array, in machine learning often called a tensor. We can then define our dataset as a relation between the input space, comprised of sets of tweets represented as a sentiment vector: X and the space of our response variable ($VXAAPL) Y.

$$D : X \times Y \text{ where } X : \mathcal{P}(\mathbb{R}^{tweet\,embedding\,size}) \text{ and } Y : \mathbb{R}$$

This dataset can be thought of as a number of samples of a function f so

$$D = (x, y) = (x, f(y)) \text{ where } f \in G : X \to Y$$

Here G is our function space, and f is our ground truth we will try to learn.

Training Task Considering the previous definitions we could say that our learning goal is to find a function f^* out of our function space so that:

$$f \equiv f^*$$

It is however very unlikely that we find this exact match, so we will have to settle for an approximation, meaning we will try to minimize the difference between f and f^* we can express this as minimizing ϵ with to a cost function C

$$\epsilon = \int_X C(f(x), f^*(x))\, dx \text{ where } C : Y \times Y \to \mathbb{R}$$

We of course don't actually have the function, instead we have a dataset D from which we will select a validation part V. Which finally allows us to write our main objective to minimize ϵ in

$$\epsilon = \sum_X C(f(x), f^*(x))$$

As cost function, further called loss function, we will mainly use the Mean Squared Error function (MSE). When training our algorithms, we will of course only give them the training set T with $T = D \setminus V$
For evaluation we will use the previously mentioned walk forward validation strategy. Since the specific setup will differ between different algorithms we will detail the approach in the relevant sections.

4.2 Dataset

One of the vital parts of every machine learning project is the dataset, in this chapter we detail the origins and construction of our dataset.

4.2.1 Provided

The provided part of the data consisted of a list of tweets about specific tickers, the one focused on was $AAPL. Every tweet came with its text, the date at which it was posted and a unique id. They were largely ordered by date. There were some peculiarities though when looking at the data it became clear that not every day had tweets, this while in the case of $AAPL there were on average more than 600 tweets per day. When looking deeper at the dates supplied in the source and comparing them to the ones online, they seemed correct, implying that the dataset does not contain all tweets posted in the covered timeframe, more specifically it seems that there are way more days not represented by tweets then would statistically be expected with even a remote assumption of uniform distribution of tweets over days. The size of our dataset is detailed in Table 41.

4.2.2 Gathering Data

While the list of tweets was provided the images still had to be acquired. This was not trivial, as it meant looking at hundreds of thousands of tweet and downloading a lot of images (in the order of tens of thousands).

Scraping

As said the provided data contained permalinks to their twitter page, so it is important to know where on a page the images can be found. In our application we will look for them inside an open graph image meta tag. These open graph tags are explicitly meant for scraping, and are used by the likes of Facebook and Twitter. Since the dataset spans millions of tweets and the scraping being very IO bound I wrote the application to use multiprocessing. Since we can acquire the image link from parsing the HTML, we don't need a full renderer which allows us to really up the number of parallel processes. Rate limiting also was not a problem.

The application consists of a controlling process, a process parsing the file with all the tweets, and a pool of 100 processes processing our tweets. This processing is straightforward. It extracts an open graph image URL, determines if it is a relevant image and if it is it downloads it. Because this means a lot of small files being written at the same time, images are saved in ram for a wile and written to a drive in large, uncompressed archives. These archives were copied to the VSC cluster and then extracted.

4.2.3 Data Preparation

Since our models don't just work on straight up text or images we have to project them in a feature space our model can then use to make a prediction. This is further called embedding as we embed the samples in a space then used by the model. A tweet will be embedded onto a vector by concatenating the results of sentiment analysis on the image and the text. We have elected not to remove retweets from the dataset as we believe a retweet to be expressions of the sentiment expressed in the original tweets and as such valuable.

Extracting Sentiment from Text

The aforementioned successful use of VADER in relevant sentiment analysis competitions motivated us to use it to evaluate the sentiment of our tweets. Another benefit of VADER is that we don't have to pay much attention to preprocessing. It doesn't require tokenisation[1] or lemmatisation and as it's primarily aimed at microblog text it handles specific microblog lexical features such as emoticons (both ASCII and utf-8 encoded), various abbreviations (e.g. LOL, WTF), non punctuation and capitalization (e.g. all caps or multiple exclamation marks) and specific slang (e.g. Uber, kinda). We also chose not to remove URLs and usernames, mirroring the dataset included with the project.
VADER uses a human curated sentiment lexicon for getting basic polarity and intensity values from lexical features, and several heuristics using syntactical and grammatical conventions to estimate sentiment of the sentence. This sentiment is expressed in four numbers: Positive, Compound, Neutral and Negative.
We apply this sentiment analysis for all tweets, which is computationally expensive and therefore multiprocessing is used here. Our system allocates a shared array and the tweets are iteratively scattered to a process pool. This brought a speed-up of more than 15 times for 30 processes.

Extracting Sentiment from Images

The second part of the embedding is the embedding of the image, and for this we used the ontology and pretrained models included in the MVSO project, this use was inspired by similar projects using preceding models[Kumar and Garg, 2019]. Applicable image sentiment analysis tools are not as commonly available as they are for the textual modality.

ANP Classification In the first step we go from an image to an ANP classification vector, meaning that the model classifies the image into one of 4032 ANPs, concepts with sentimental relevance in the image. For this we use a pre-trained convolutional neural network from the MVSO project.

[1] When working with longer text samples sentence granularity tokenization is advised.

Emotion Embedding After this step we have for each image a vector indicating how strong each ANP was detected in the specific image. Now we need to convert these vectors into some usable form of sentiment for our network. The route we take here is to use a list of emotional intensity for every ANP that was a part of the ontology. A naive idea here would be to select the highest ANP and use that as emotion. This however lead to worse learning performance, we opted to select a number of ANP's and to for every emotion select either the average or max value for of all selected ANP's. The emotional embedding is performed to 24 dyads from Plutchiks wheel of emotion[Plutchik, 1991].

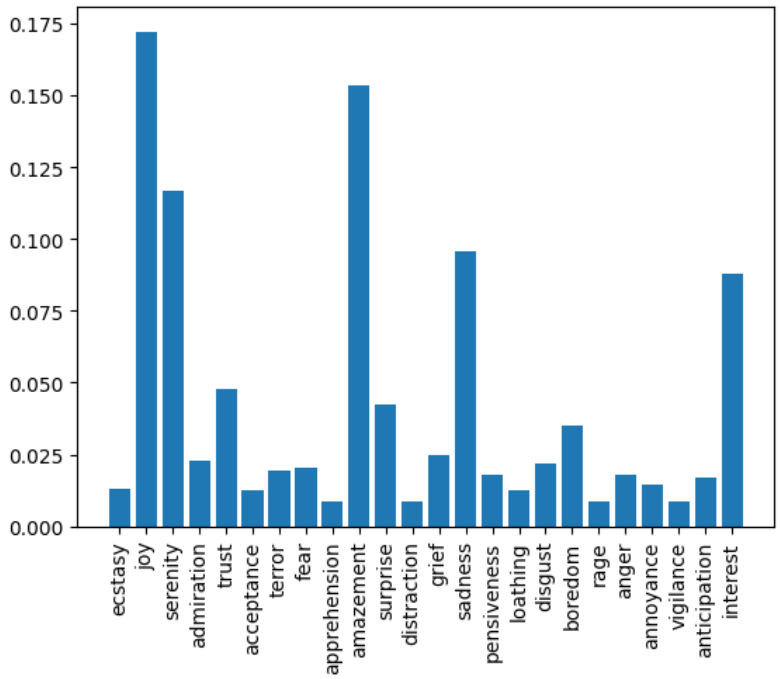


FIGURE 41: The embedding of the ANP $fresh\_flowers$ in the 24 emotions according to the MVSO

Data Cleaning A significant amount of the images are not very relevant, a lot of them being graphs, the detection of these tweets is however nontrivial and is very hard without an annotated set of examples relevant to our data. The use of chart classification tools used in the extraction of data from graphs [Savva et al., 2011] was considered but did not provide significant results. We did decide to remove days (or weekends) that did not have more than 15 tweets.

4.2.4 Response Variable

Our main response variable is the implied volatility of the $AAPL stock over 30 days. This variable is calculated by CBOE according to the VIX methodology and

sampled at the end of the trading day.

VIX Calculation Our response variable data was acquired from the CBOE website, but we'll summarize the calculations. This part is taken from the CBOE VIX white-paper [Exchange, 2009] with some modifications.
The central equation for volatility used is:

$$\sigma^2 = \frac{2}{T}\sum_i \frac{\Delta K_i}{K_i^2} e^{RT} Q(K_i) - \frac{1}{T}\left[\frac{F}{K_0} - 1\right]^2$$

Where:

σ	$\frac{VIX}{100}$
T	Time to expiration (in years)
F	Forward price level, derived from option prices
K_0	First strike below F
K_i	Strike price of i^{th} out-of-the-money option; a call if $K_i > K_0$ and a put if $K_i < K_0$; both put and call if $K_i = K_0$.
ΔK_i	Interval between strike prices – half the difference between the strike on either side of K_i: $\Delta K_\mathrm{i} = \frac{K_{i+1} - K_{i-1}}{2}$
R	Risk-free interest rate to expiration
$Q(K_i)$	The midpoint of the bid-ask spread for each option with strike K_i

These values are calculated for 2 sets of options with expiration dates between 23 and 37 days away, near-term and next-term. Every week the selection of options is rolled. The VIX is then calculated as 100 times the square of the weighted 30-day average of the two values:

$$VIX = 100 \times \sqrt{\left\{T_1\sigma_1^2\left[\frac{N_{T_2} - N_{30}}{N_{T_2} - N_{T_1}}\right] + T_2\sigma_2^2\left[\frac{N_{30} - N_{T_1}}{N_{T_2} - N_{T_1}}\right]\right\} \times \frac{N_{365}}{N_{30}}}$$

Where:

N_{T_1}	number of minutes until the settlement of the near-term options
N_{T_2}	number of minutes until the settlement of the next-term options
N_{30}	minutes in a month
N_{365}	minutes in a year

4.2.5 Dataset Organization

As our code ran on the Genius cluster system memory was not really a constraint. The dataloaders run in multiple python threads constrained by the Global Interpreter Lock,

this prevents true concurrency but allows fully transparent access to memory without need for extensive synchronization. This meant that the practical approach was to put all information that could be relevant in memory, to improve the performance of the data pipeline.

Logical Organization The data is organized chronologically, every tweet being embedded into a vector $\in \mathbb{R}^{28}$. The 24 first features are the dyads extracted from images by summing the emotional embeddings41 of the 5 ANP's with the strongest activation. The last 4 are the polarity scores extracted with VADER as described previously. The data is presented as an array of vectors $\in \mathcal{P}(\mathbb{R}^{28})$ being the embedding of all tweets on a certain day. Weekends are combined into a single day, bank holidays are not considered. Data is of course only used if a matching response sample is available. The response variable was shifted by one day. Properties of the features and their relations are documented in chapter 5.

Tweets	555056
Tweets Multimodal pre cleaning	71222
Tweets Multimodal post cleaning	**56402**
Days pre cleaning	830
Days post cleaning	**658**
Startdate	2012-12-31
End Date	2019-09-11
Shape of date presented to neural networks	$658 \times sample\_size \times 28$
Shape of data presented to 'naive' algorithms	56402×28

TABLE 41: Our dataset in numbers

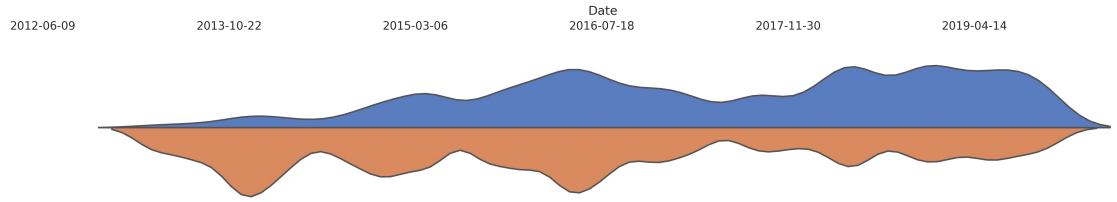


FIGURE 42: Distributions of tweets in time, the upper area represents the cleaned, multimodal tweets, the lower area represents all the tweets. Areas are not to scale

4.3 Naive Application of Machine Learning

Before using complex architectures it's valuable to use more simple models to study learning behaviour and to assess the performance of further models. The algorithms we use for this are XGBoost and linear regression, popular machine learning tools. We start here to assess the performance of a more naive approach and to justify our more complex models later.

Naive vs Aggregating When trying predictive models in settings with similar unordered, variable sized inputs the common approach is often to aggregate all samples into a single fixed dimension vector using hand crafted features (e.g. mean, max, stddev). This eases implementation and provides robustness against noise and bad data. We however train the models to predict tweet by tweet predictions which will be aggregated (mean) per day for evaluation. We decided against the common approach for a couple of main reasons:

- We only have a limited number of datapoints (+- 600) but we have a much larger number of tweets (+- 70k), so we hope that our models can therefore learn better when exposed to the much larger amount of data.
- The tweets are far from evenly distributed over days, which makes the use of a lot of hand crafted features difficult.
- A model trained on a tweet by tweet granularity may be useful when predicting IV on a real time basis.
- The models are presented in contrast to further deep learning models that are able to consume samples without external aggregation. In their simplest form they will resemble the mean of individual predictions strategy used here.

4.3.1 Prediction & Evaluation: Setup

Our dataset D logically has the following structure presented to the training loops:

$$D : X \times Y \text{ where } X : \mathbb{R}^{28},\ Y : \mathbb{R}$$

and its content is a number of samples of a target function f:

$$D = (x, f(x))\ x \in X \text{ where } f : X \to Y$$

so that our models will select a function f^*:

$$f^* : X \to Y$$

to minimize n objective

$$\epsilon = \int_{D_X} C(f(x), f^*(x))\, dx$$

which it could calculate as follows:[2]

$$\epsilon = \frac{1}{\#samples} \sum_{X_{train}} (f(x) - f^*(x))^2$$

Practically speaking D is a Matrix of shape $(samples, features)$ with in our specific tasks 28 features and around 70k samples.

Evaluation For a robust evaluation of performance on the whole set we will use multiple runs with different datasets, this means we will evaluate models by the metric

$$\epsilon_{ev} = \frac{1}{\#S} \sum_{S} \left(\frac{1}{\#s_{val}} \sum_{s_{val}} \left(f(d) - \frac{\sum f^*(s_{train}, x)}{\#x} \right)^2 \right)$$

Here S are the splits we evaluate over:

$$\forall s \in S : s = train, val \text{ so that } s \vdash D$$

Embedded tweets x are organised in days:

$$d \in days : days \vdash s_{val}$$

Because a normal cross validation would be a non-realistic situation we will evaluate the model on the 5 last folds of a 10-fold walk forward cross validation.

4.3.2 Linear Regression

Linear regressions are some of the simplest machine learning algorithms. They have no hyperparameters, and are deterministic, we ran a linear regression over the dataset as previously presented.

4.3.3 XGBoost

Gradient Tree Boosting was the second algorithm we evaluated. It was allowed train for 200 rounds but early stopping stops training when the RMSE stopped decreasing for more than 10 rounds. We use the dart booster [Rashmi and Gilad-Bachrach, 2015] which means that more than one tree will be trained every round, depending on the number of trees dropped in a round, which allows us to reduce the effect of overspecialisation, and let the later training stages contribute more to predictions.

Tuning We tuned the depth, eta, gamma and dropout rate hyperparameters. The large amount of feature engineering and dimensionality reduction was shown in the optimization favouring relatively low tree depths. It is here that the value of Bayesian optimization becomes clear, we sampled $\pm$ 1000 configurations, which means that if we performed a grid search less than 6 values per variable would have been possible.

[2] This is a simplification, sometimes smoothing is used for better training performance, other losses might also be used

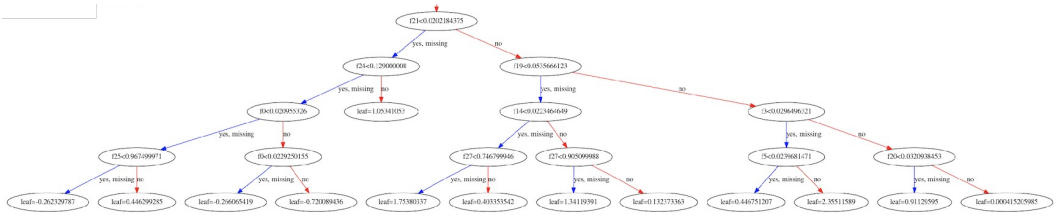


Figure 43: A part of a decision tree

The acquisition function of the optimization was expected improvement, with one random sample point every 10 evaluations to limit the greediness of the search.

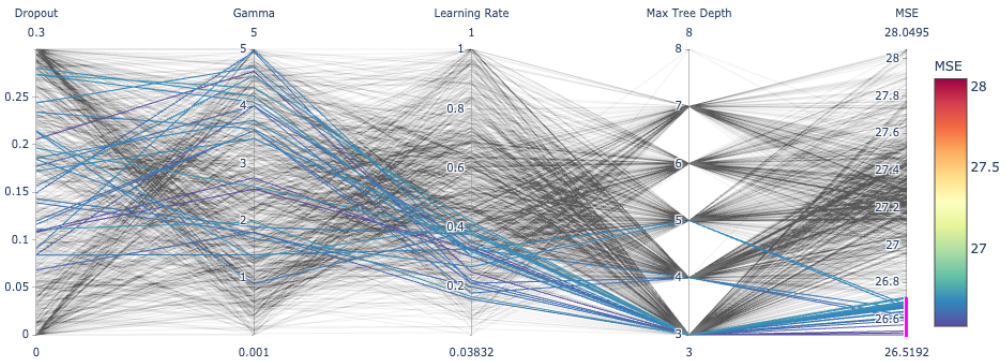


Figure 44: A parallel visualization XGBoost tuning results, the 25 best performing configurations are highlighted. It is clear that the model is selecting a conservative configuration, indicating a noisy dataset.

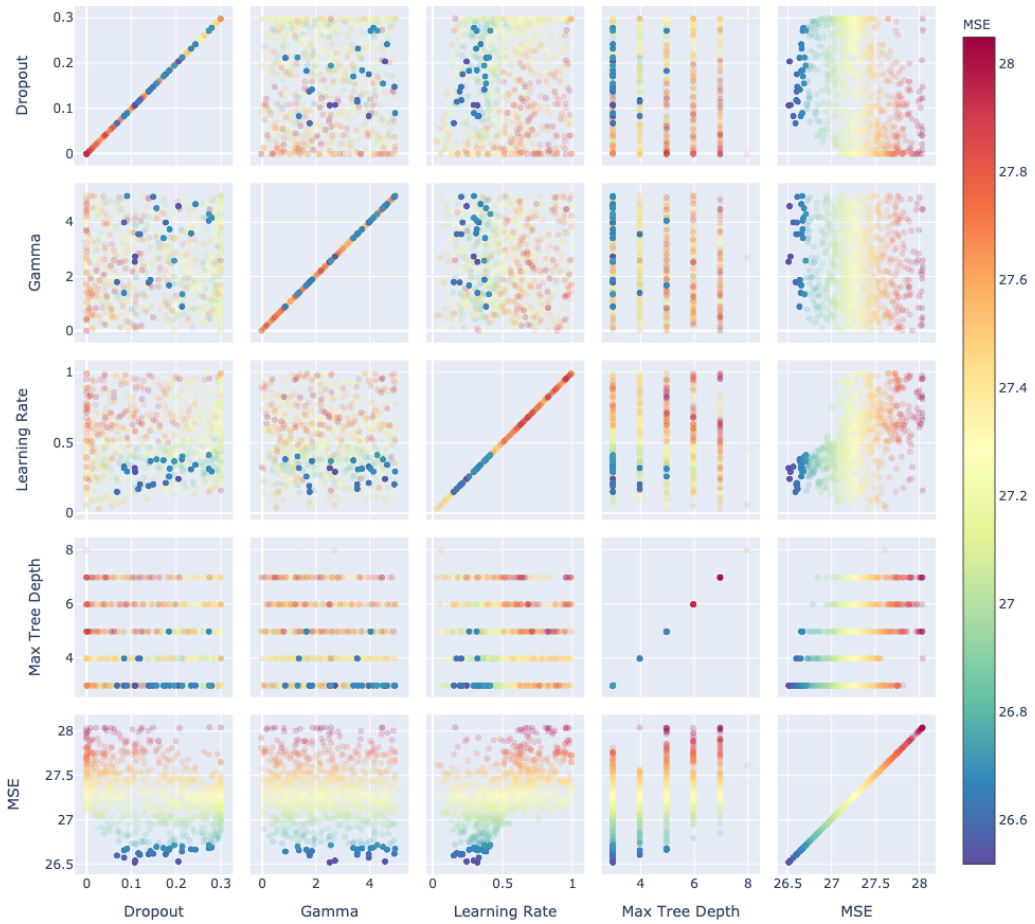


FIGURE 45: An alternative visualization of XGBoost, tuning results the 25 best performing configurations are again highlighted.

4.4 Deep Learning Architectures

In this section we will use neural networks to fit our prediction problem. Contrary to preceding experiments we will be using the whole day to train networks, instead of just single tweets. This means our models will be trained on an unordered sequence of vectors, each one representing a tweet posted on that day. For this we will use Convolutional Deep Averaging Networks, as proposed by [Gardner et al., 2019]. After this we will propose some modifications to these networks to improve performance.

4.4.1 Learning Goal

Our dataset D has the following structure:

$$D : X \times Y \text{ where } X : \mathcal{P}(\mathbb{R}^{28}),\ Y : \mathbb{R}$$

and its content is a number of samples of a target function f:

$$D = (x, f(x))\ , x \in X \text{ where } f : \mathcal{P}(\mathbb{R}^{28}) \to \mathbb{R}$$

so that our models will select a function f^*:

$$f^* : X \to Y$$

to minimize an objective

$$\epsilon = \int_X C(f(x), f^*(x))\, dx$$

which it could calculate as follows:

$$\epsilon = \frac{1}{\#X} \sum_X (f(x) - f^*(x))^2$$

Evaluation For a robust evaluation of performance on the whole dataset we will use multiple runs with different datasets, this means we will evaluate models by, among others, the metric:

$$\epsilon_{ev} = \frac{1}{\#S} \sum_S \left(\frac{1}{\#s_{val}} \sum_{s_{val}} (f(x) - m_h^*(s_{train}, x))^2 \right)$$

Here S are the splits we evaluate over:

$$\forall s \in S : s = train, val \text{ so that } s \vdash D$$

and m^* a deep learning model:

$$m^* : \underbrace{H}_{\text{hparams}} \to (\underbrace{\mathcal{P}(X \times Y)}_{\text{training set}} \to \underbrace{(X \to Y)}_{\text{function inference}})$$

Which in our case will be

$$m^* : \underbrace{\mathbb{R}^{\#hparams}}_{\text{hparams}} \to (\underbrace{\mathcal{P}(\mathcal{P}(\mathbb{R}^{28}) \times \mathbb{R})}_{\text{training set}} \to \underbrace{(\mathcal{P}(\mathbb{R}^{28}) \to \mathbb{R})}_{\text{function inference}})$$

4.4.2 Basic Architectural Concepts

High Level Approach

Deep learning is representation learning, models learn successive representations of their input with the last being the output. Every layer defines a transformation into a new representation, for our model to be applicable we will need to construct it into a way that it:

1. Is able to represent the transformation from input to output.

2. Can learn the transformation from input to output.

One thing challenging about our specific problem is that the input has a varying dimension whereas the output has a fixed dimension, the number of tweets posted on a day differs greatly and it is the size of this *sample_size* dimension that will vary significantly over different days. This necessitates[3] a transformation from an unknown dimension to a fixed dimension somewhere in the model. Our models will always have three distinct parts to them: An embedding part, that keeps the amount of samples and works on a per sample basis, an aggregation part that converts the per sample representation into one representation representing the full sequence of tweets into a fixed size vector, and lastly the regression part that uses this global representation for the final prediction.

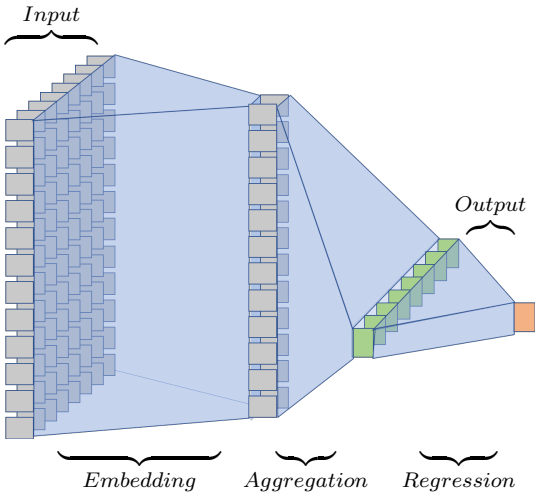


FIGURE 46: The three parts of a CDAN model

[3]There are other workarounds but they either are tough for our specific problem (RNN/LSTM, Zero-Padding) or would reduce the true power of our model (Constant Size Sample Binning)

Embedding

The first part of all our models will perform an embedding. This means we will learn an embedding function for the sentiment vectors that represents them into a machine interpretable space, learned by our model. It is implemented by one or more one dimensional convolutional layers with the size of the input at that level as filter size, by adding more filters we can increase the dimension of the output embedding. In our models we apply at least one convolution layer over the data, this layer generally has a kernel size of 1 and n filters. It will transform our data with the shape $(1, 28, sample\,size)$ to $(1, n, sample\,size)$. The convolution here only operates on a single sample at the same time, and does not consider different samples. The embedding part is[4] therefore isomorphic to the application of a fully connected network onto the samples one by one. This mirrors the approach taken in the previous experiments, but whereas the XGB and Linear Regression were trained to regress towards an IV value we will here be able to regress towards a representation used by the final part for the final regression. It's in this concept we believe the power of our networks lies, the embedding function is able to be learnt over a large number of samples, while the final regression is the only part constrained by the low number of dates.

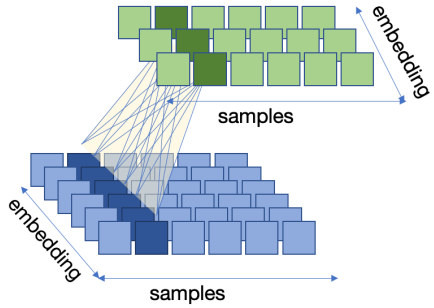


Figure 47: A 1d convolutional layer is isomorphic to the application of a fully connected layer over the feature channels of samples.

Aggregation

After the embedding stage we still have a dimension of varying size along our sample size, stemming from the varying amount of tweets posted in a day. To make a prediction we will have to transform this representation into one with a known dimension. For this we will use an adaptive pooling operation that will generally perform a transformation with the following shape $(1, embed\,size, samplesize) \rightarrow$

[4] in its simplest form, subsequent proposed modifications violate the assumption of independent treatment of tweets deeper in the network which causes the property of permutation invariance along the sample axis to be invalidated

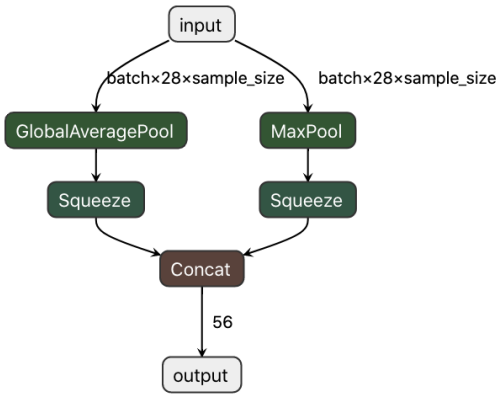


FIGURE 48: Example of an aggregation operation[5].

$(1, embed\,size, 1)$,often squeezed to $embedsize$. In most cases we performed max and avg pooling and concatenated the result.

Post-processing

After the aggregation is applied we have a tensor of constant shape, this allows us to use a classic neural network to make our final prediction. The actual size of the tensor depends on the setup of the network but generally it will have $(1, 1, 2 \times embedding\_output\_size)$ as shape. Here $embedding\_output\_size$ is the output size of the embedding function of the model, often the nr of kernels in the last convolutional layer. In most cases we will use a hidden dense layer and a dense output layer here. These layers are the ones actually making the prediction. We strive to keep the size of this part as small as possible.

4.4.3 Architectural Approaches

Here basic concepts of evaluated network topologies are explained, specific structure, graphs and size can be found in the appendix.

Pre Net

A first, naive approach could be to not post process the information and instead fully rely on the embedding to preform the predictions, and taking the mean of the predictions. This is a relatively simple methods that removes all interaction between samples. This approach closely mirrors the XGB approach, in this network we used Residual style skip connections, which allows us to deepen the network without having gradient issues.

[5]Due to the varying size of our sample size our batch_size will always be one, this is why it is squeezed away.

Post Net

The inverse approach is to perform no embedding and aggregate straight away, after which we perform the prediction. This approach is in line with traditional fixed granularity sampling for multiple instance learning. In our cases it however performed poorly with bad and unstable results, even for small network sizes. This leads us to conclude that this approach was not worth exploring further. It also reinforces our feeling that the CDAN structure is valuable in this application. We believe that the reason for the problems in this approach lie in the small number of datapoints, and while the CDAN networks are ostensibly also limited by this, they can learn their embedding function over a much larger set of samples.

Mixed Net

The logical next step is to combine the two approaches, these models perform a lot better than the previous ones, and are implementations of the CDAN architecture. The first implementation was very simple, consisting of one embedding layer and two linear layers, which clearly limited its performance.

Pooling Net

A main limitation of our method is that the interaction between samples is very limited, we can try to allow this by letting the model sample features from other local samples with pooling, as demonstrated in Figure 49. It's important to note that this violates some basic properties of the CDAN network, mainly the permutation invariance towards the input, since the pooling operation is locally applied. To avoid our networks learning spurious spatial patterns we shuffled the samples every epoch.

AttNet

In the previous model we only evaluated part of the samples when exploring interactions, but it could also be interesting to evaluate the whole input. If we were tackling this problem with statefull networks, this would be analogous to Attention mechanisms. The model can attend to the state previous of the current layer for every sample.

Residual Blocks

During the evaluation of Pre Net the performance of the residual blocks was pretty convincing. This pushed us to also implement residual blocks in our more complex models. We implemented a residual block for the Pooling Net since it was the best performing model. We also used it on mixed net.

Leveraging Monomodal Data

Until now we have always used only the tweets that were accompanied by an image. This however is only a subset of the total dataset, therefore we would like to use that

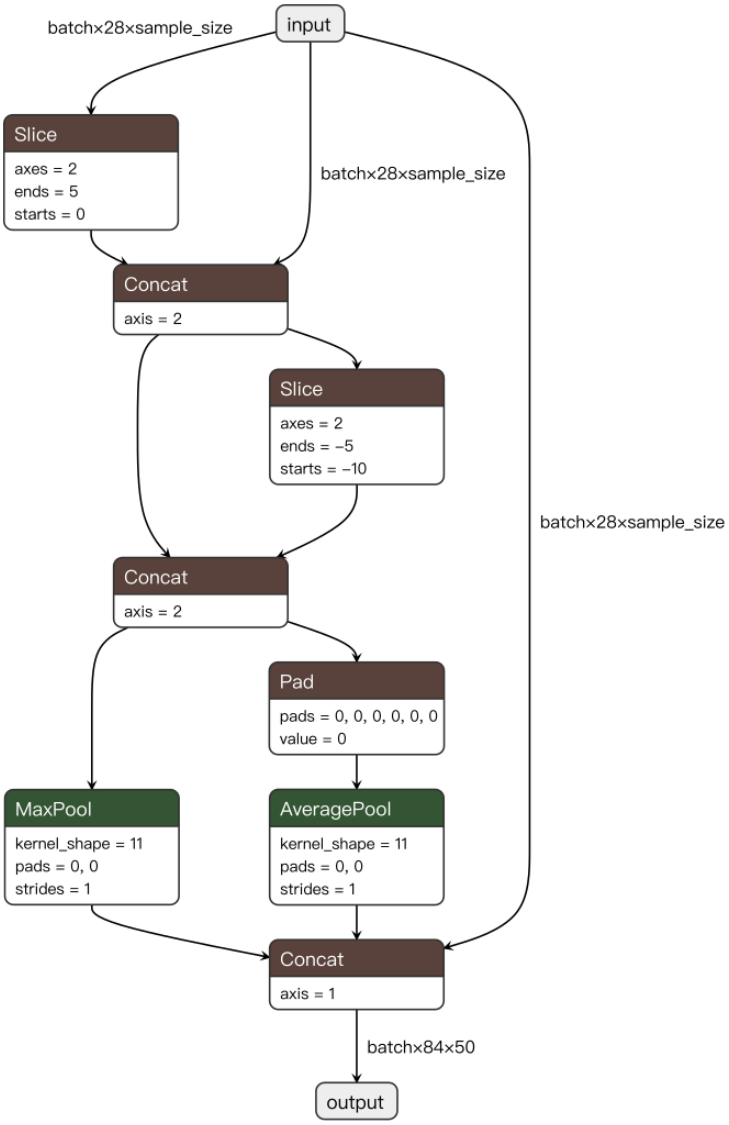


FIGURE 49: Example of a pooling operation

information in our architecture without losing our multimodal focus.

Parallel Embedding The first approach used is to perform a similar embedding and then aggregate, we then combine it with the result of the multimodal aggregate. This lets our post processing part look at the two different parts for making its prediction.

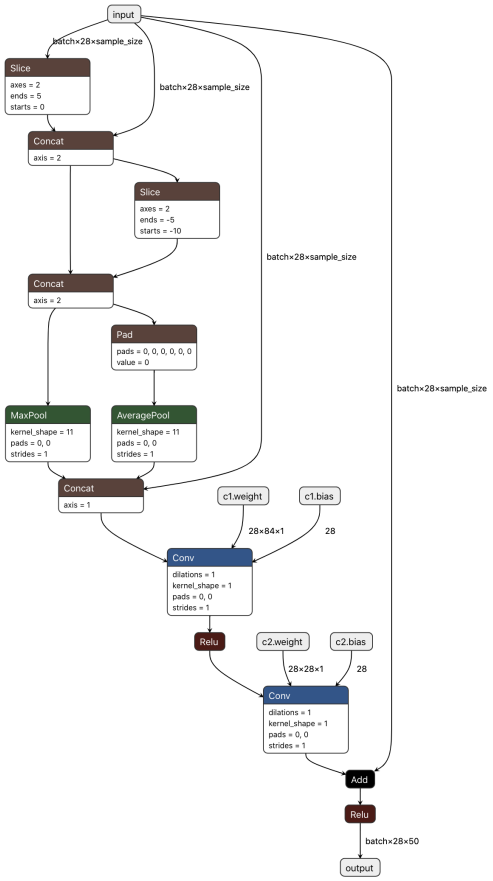


Figure 410: Implementation of the Residual Block in the Residual Pooling Net

Distributive Pooling In our Pooling Net model we let the model inspect the state of other samples while embedding, it would of course be useful to also be able to inspect some state of the tweets. This is however a lot more complicated. The amount of tweets without images is quite a lot bigger than the amount of tweets with images. This forces us to distribute the text embeddings among the tweets with text. For this we again will use a distributive pooling strategy.

Leveraging More Data

Due to our rather limited amount of data our results will likely not be very competitive with models used in production. This is to be expected as our models focus on the multimodal part and its interaction with the CDAN concept, therefore they only consider the data coming from the tweets. Models used in productions will not have those limitations. To quickly evaluate the performance of our model when given more fundamental information we allowed our model to inspect the previous 4 days when making its prediction. When a day was not available, we replaced it with the

mean. As it is the best performing model we injected the data into Pooling Net Res 5 after the aggregation stage.

4.4.4 Learning and Evaluation

Training The networks were trained on CPU with double precision, mini-batch size was one because our input had variable size. We used Huber loss and MSE loss, and used stochastic gradient descent with momentum and Adam[6] as optimizers. Because our batch size was one, we implemented gradient accumulation to improve learning behaviour.

Evaluation Due to deep learning being computationally much more expensive than the other methods we were not able to fully evaluate the impact of all possible variations. For all evaluations we again used the 5 last folds of a 10-fold walk forward validation. A full validation of a similar network took up to 10 hours for some configurations but was often in the range of 2 hours. Training on GPU was tried but did not significantly improve training speed. These networks are not that large and the dynamical axes and the batch size being 1 are the main drivers of this disappointing performance.

Multiple Loss Layers

For some models we added a regressing convolutional layer before the aggregation to perform a sample specific prediction, this allowed us to return that value and get a stronger gradient which could help the embedding to perform better. This is solely for training purposes and this output is not used in evaluation. This approach did not improve generalization significantly but accelerated early learning phases.
The losses recorded in this training were, when averaged over the full day, a bit worse than the whole CDAN, this again is an indication that our final regressing part is useful. The fact that final regression generalization performance was not impacted seems to support our idea that the individual samples are all fully used in the learning of the embedding function. This application of the network can be seen as the batched application of a normal fully connected neural network to the different tweets.

[6] we also evaluated the strongly related AdamW optimizer

Chapter 5

Results

In this chapter we will discuss results of our experiments. It's important however to understand that this project concerned mostly the predictions from multimodal sentiment, and the CDAN architecture we use to predict IV. The architecture and approaches are therefore implemented with an important constraint, namely that the multimodal input should be the driving input data. Our first section will show the results of our dataset construction We will then evaluate different models against each other, only giving them access to the multimodal sentiment as inputs, we will discuss choices made in training, how they affect performance, and how the models behave when we look deeper at our way of validating their performance. We will finally take our best performing architecture and give it access to previous IV values, this will allow us to more fairly compare it to a more practical baseline: the lagged values of the IV.

5.1 Dataset Exploration

Its impossible to understand what our models might do without understanding how our data actually looks. So we'll start with looking at our dataset.

5.1.1 Quality

The first insight in the data is immediately a reality check. People don't write tweets to be understood by machines, nor do they convey clear sentiment in the images they add to their tweets. A lot of the tweets are very referential, using a specific set of words and phrases. The images are not all relevant, and a lot of them are graphs of the stock. When looking at the data sample for sample it's not easy to imagine accurate predictions, but we can't forget that we will not be making predictions based on single samples.

5.1.2 Embedding

Our multimodal extraction steps embed the tweets in a 28 dimensional space, here we will attempt to visualize the characteristics of these dimensions and how they

relate to each other, first we will demonstrate the distributions of these variables.

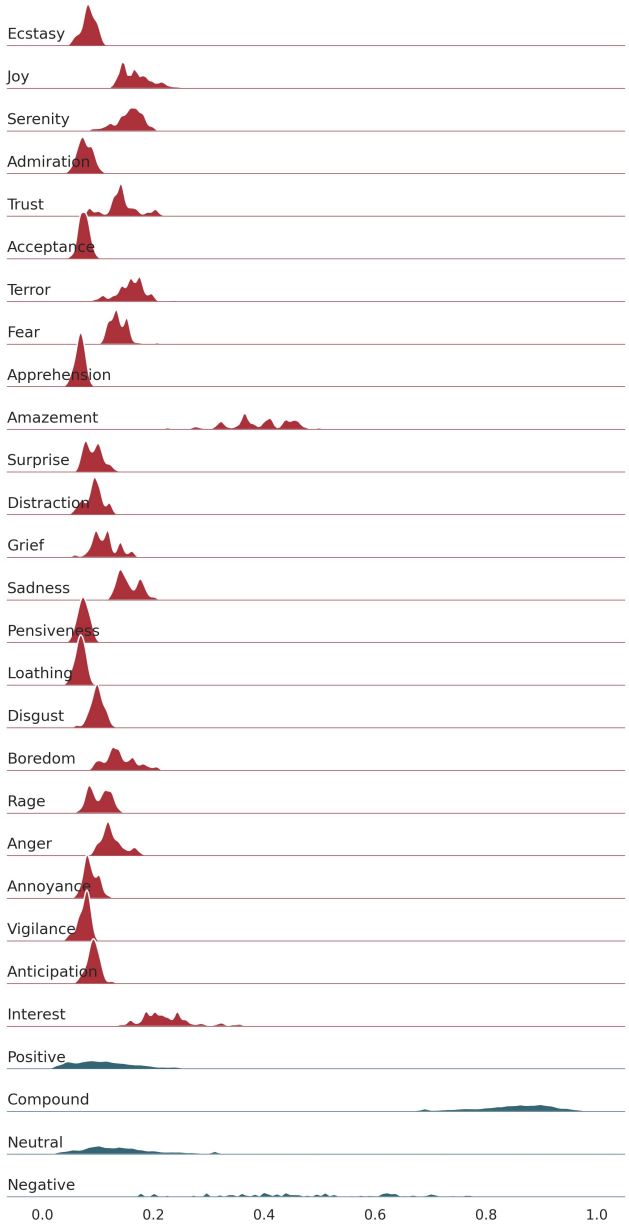


Figure 51: Probability distributions (KDE) of the dimensions, the change in color denotes the change in modality. Samples drawn from the dataset as presented to models, with tweet-level granularity

Distributions of Features Looking at Figure 51 it's clear that the visual embedding values are largely concentrated in the earlier parts of the $[0, 1]$ interval, and this

way more than the textual features. We have experimented with normalization of the features but as this did not seem to impact performance positively we did not include this in our final evaluated methods.

Relations Between Features After the distributions it's also relevant to look at how the features relate to one another. To investigate this we calculated the correlation between them. Here we would like for the absolute value of the correlation to not be too high, as that would imply low individual merit of the features, but it being zero would be problematic as the features represent emotions that should somewhat correlate. We visualize the correlations between features in Figure 52. Our features do not correlate too much, but it's likely that the signal could be

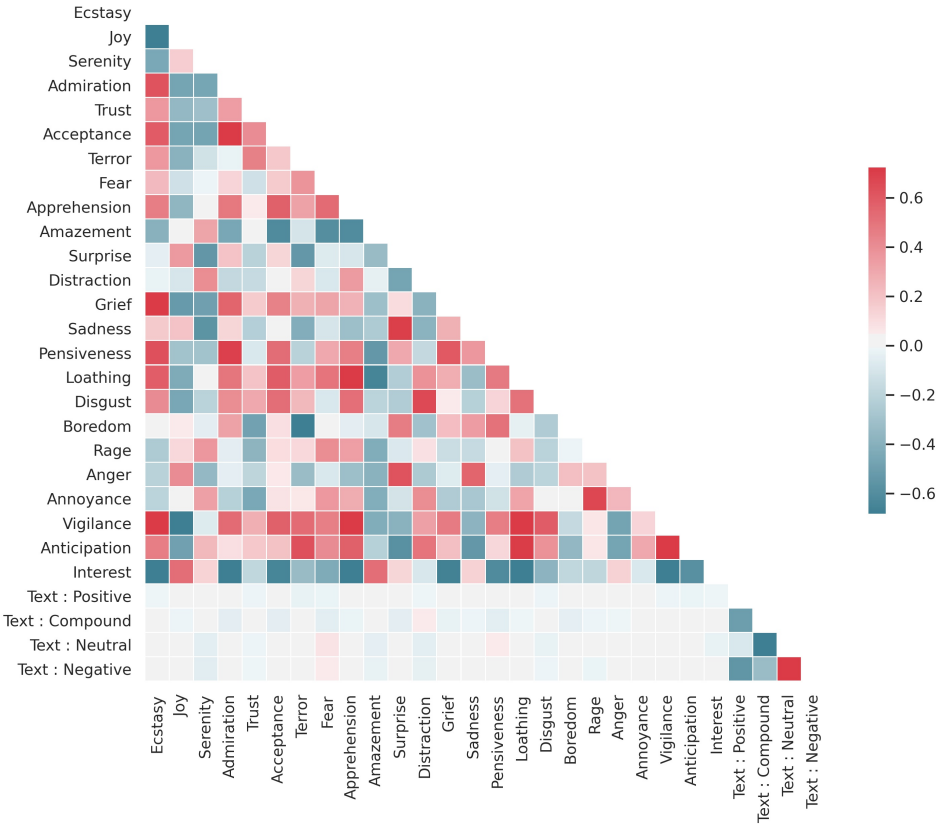


FIGURE 52: Correlation between features. Samples drawn from the dataset as presented to models, with tweet-level granularity

compressed/downsampled without losing much information. This is however not trivial, and due to the extensive tweet level granularity representation layers of our models we chose not to dig further. PCA's of the input data also were not stable across different subsets of the data. More problematic is the very low correlation between the two modalities, which in a problem agnostic view would be optimal, as

correlation between features is usually avoided, but in our case it implies that the text sentiment extraction and the visual extraction don't agree on the results, while we would expect some correlation between sentiment expressed in images and the accompanying text.

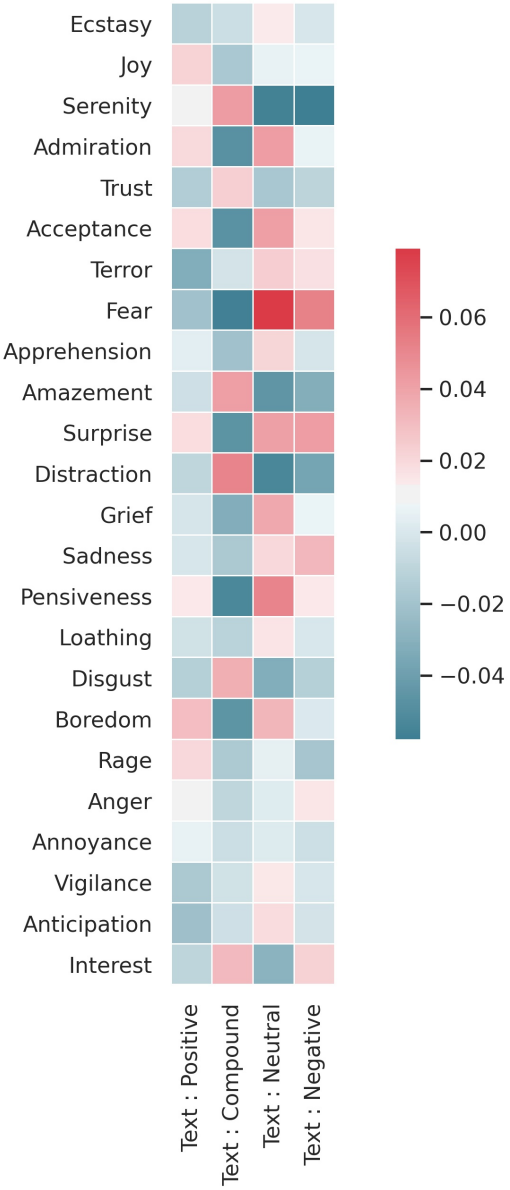
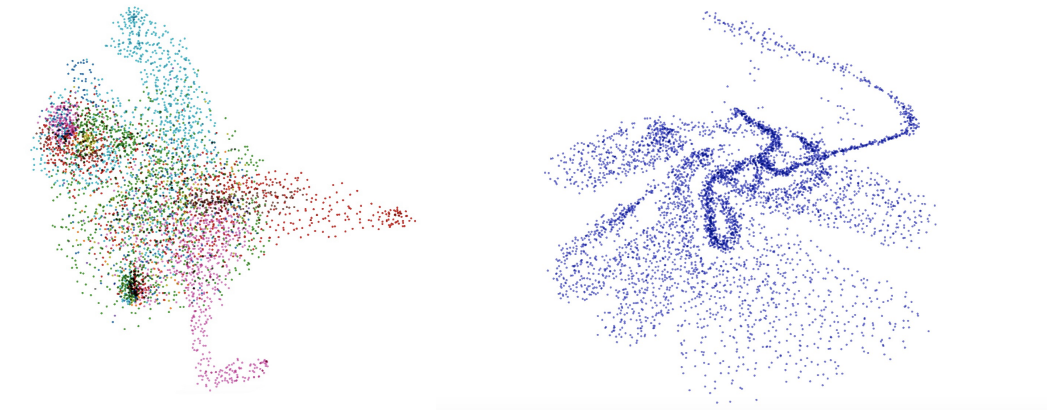


Figure 53: Correlation between features, focused on the correlation between the textual features and the visual features. Samples drawn from the dataset as presented to models, with tweet-level granularity

UMAP & tSNE UMAP and tSNE are algorithms that try to reduce the dimensions of points in a high dimensional space, they are most often used to visualize high dimensional datasets or embeddings generated by hidden layers in deep learning networks. These visualizations are a bit of a dark science. When trying to represent the emotions, we looked at different ways of sampling the emotions. The structures shown by the algorithms are very different, and when binning the response variable into 25 bins there is no clear structure relating to the value. tSNE did generate diagrams that seemed to show response correlated structure sometimes but these did not seem to be significant.



(a) Generated by tSNE (b) Generated by UMAP

FIGURE 54: Examples of representations generated by dimensional reduction algorithms

Usefulness of Features In the case of deep learning it is often very hard to measure the importance of input features. XGB however offers us a metric called the F-score, which is the number of times each feature is used for a split in a tree. While not perfect this score offers an indication if some features are ignored by the model. We plotted these sores in Figure 55. It's clear that our algorithm is pretty reliant on single variables of textual sentiment but in aggregate more attention is given to the visual features. This seems to indicate that there are no features that are clearly useless. The combination of the previously mentioned low correlation between modalities and the use of features stemming from both text and visual sentiment reinforces the idea of the model relying on both.

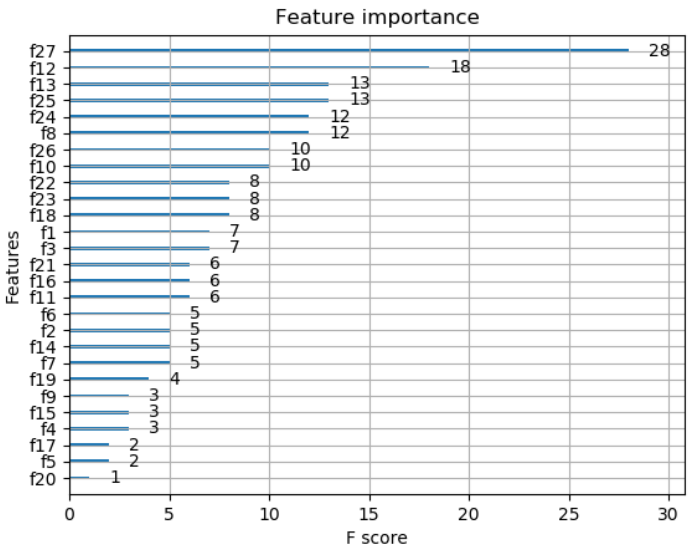


Figure 55: Diagram indicating importance of features for XGBoost, f24-27 are sentiment extracted from the textual modality. F-score is only an indication of feature importance. These results were relatively stable across evaluations however, and consistent with other, related metrics such as average cover and average gain

5.2 Learning Performance

In our research we used a lot of different architectures. They can generally be split into two approaches, In the first we try to predict the IV for every single tweet and then average that. In the second we try to leverage the amount of data and perform the aggregation in model. Some of these models also have mechanisms to let the samples interact.

5.2.1 General Performance

To accurately evaluate our models we will mainly use the MSE over the predictions per day. In our figures the RMSE is simply the root of the MSE which is shown as it is in the same size range as our predictions. We also show the MAE, which is less sensitive to outliers. It's clear that our models, which can leverage the full sample of tweets, are often better than the models only using final stage aggregation. The models that allow for interactions between samples during the embedding stage perform even better. The simplicity of Mixed Net (1 convolution and 2 fully connected layers) is clearly limiting its performance, implying that there is definitely a benefit to a more complex architecture on the embedding size. This is confirmed by Pre Net, without post aggregation layers performing really well, owing to its residual like structure. It was the performance of this residual embedding networks that pushed us to look at giving more models a similar residual embedding.

	MSE	MAE	RMSE
Linear Regression	33.607	5.791	4.888
XGBoost	26.511	4.319	5.148
Mixed Net	29.000	4.347	5.141
Res Pre Net	23.119	3.898	4.685
Res Mixed Net	21.260	3.792	4.521
Att Net	26.338	4.183	4.980
Pooling Net	21.219	3.633	4.496
Pooling Pre Res Net 5	22.085	3.718	4.591
Pooling Res Net 5	**19.4516**	**3.473**	**4.157**
Pooling Res Net 20	23.445	3.837	4.847
Pooling Net Plus[1]	19.692	3.567	4.302

TABLE 51: Out of sample performance of the considered approaches

5.2.2 A Deeper Look into a Validation Run

When looking deeper into the validation, we can see that the specific results of each fold are pretty different. The 4th fold for example often has an MSE much higher than other ones, as demonstrated in Table 52. The last folds also often have the best MSE. We decided to look deeper into the performance in these last folds. When using more, and thus smaller, folds this variability becomes clear, as illustrated in Figure 56.

Fold	1	2	3	4	5
MSE	16.872	14.378	19.769	41.329	13.7487
VAE	3.262	3.022	3.541	4.993	3.349
RMSE	4.107	3.792	4.446	6.429	3.707

TABLE 52: Out of sample performance of Pooling Net

5.2.3 Choices in Training

When performing our deep learning experiments we made a couple of choices, because of the computational expense we could not verify these choices as thoroughly as with the XGBoost parameters. The decisions were mainly based on intuition, experience and some experiments. Here we'll demonstrate some of these choices and their effects.

Huber vs MSE We used Huber loss (also called smooth L1 loss) as MSE often seemed unstable in late stages of training. This very likely is caused by the sensitivity of MSE to outliers due to its quadratic error term.

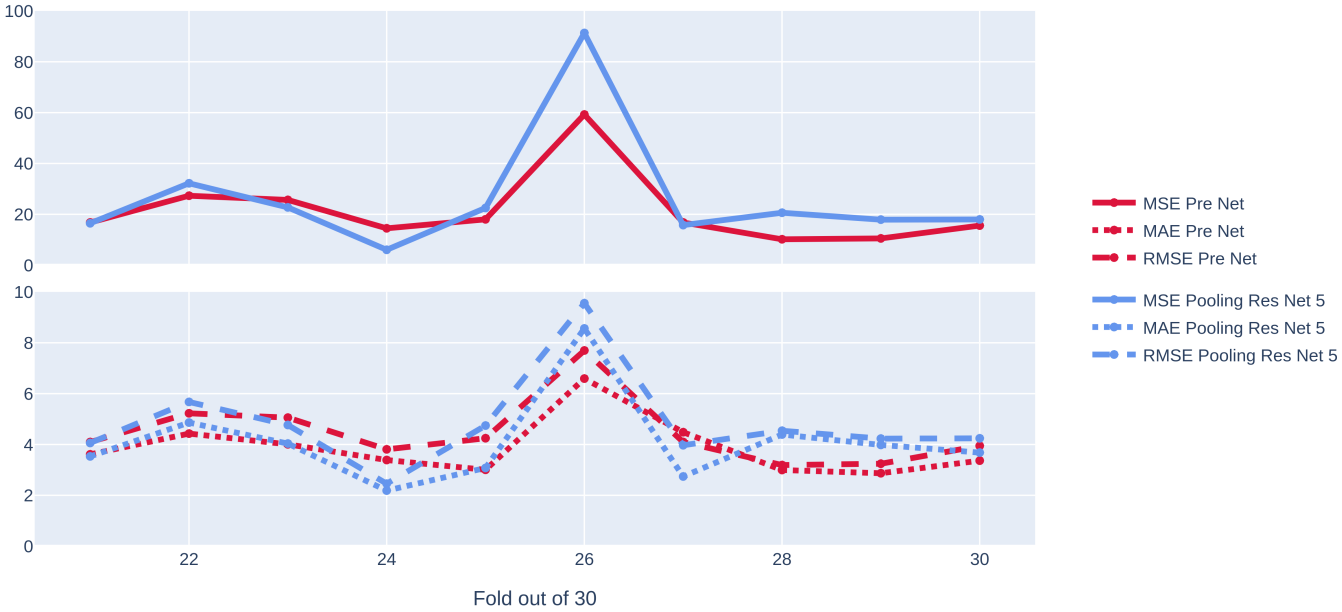


Figure 56: Out of sample scores for the last third of a 30 fold walk forward validation

Pooling Net Res 5			
	MSE	MAE	RMSE
MSE loss	23.294	3.807	4.651
Huber loss	19.4516	3.473	4.157
Pre Net Res 5			
	MSE	MAE	RMSE
MSE loss	25.506	4.092	4.893
Huber loss	23.119	3.898	4.685

Table 53: Comparison of performance for different losses used during training.

Adam vs SGD Another important choice is the choice of optimizer, while Stochastic Gradient Descent is often considered the default, Adam is also a very popular choice. The differences here were not always very significant in performance but the time to convergence was lower with Adam. This is consistent with common understanding of these algorithms. Sometimes SGD with momentum is said to be better at generalization but this was not very evident in our tests. AdamW, an adaptation of Adam with a modification to its regularization was also tested but found not to provide results significantly different from Adam.

Pooling Net Res 5			
	MSE	MAE	RMSE
SGD (momentum)	21.170	3.558	4.444
Adam	19.4516	3.473	4.157
Pre Net Res 5			
	MSE	MAE	RMSE
SGD (momentum)	23.754	4.092	4.873
Adam	23.119	3.898	4.685

TABLE 54: Comparison of performance for different optimizers

5.2.4 Realistic Performance, Comparison with Lagged Baseline

When looking at the preceding scores we can see that the accuracy is not good enough to be used in production environments, as it would not even beat the baseline of a lagged model. But this project focuses on the multimodal aspect of our dataset, and the architecture of the models reflects this. A model used in production would also leverage a lot of more conventional information. To evaluate our architecture in a similar situation we implemented a model that considers a small bit of this information. The model received the response variable for the 4 preceding dates, if they exist. Missing dates are represented by the mean. This dataset is incomplete and should be restructured if it was to be used in this manner, since this experiment is merely to indicate predictive power of these architectures when considering more data this was considered outside the scope of this project. In Table 55 we show some results, where the metrics of the models are measured again as the mean of out of sample scores for the last 5 folds of a 10-fold walk forward validation. The scores for the datasets are calculated on the whole set, as it is presented to the models. These numbers do not compare the exact same things and should be therefore be interpreted as indicative of possible performance.

	MSE	MAE	RMSE
Pooling Net Res 5 with previous IV's	8.522	2.172	2.822
Pooling Net Res 5 without previous IV's	19.4516	3.473	4.157
Presented Dataset	13.720	2.350	3.704

TABLE 55: Predicting IV with recent history available.

Chapter 6

Conclusion

In this final chapter we will cover the two main goals of this project and discuss our achievements in relation to them. We'll also discuss possible future improvements and applications.

6.1 Achievements

6.1.1 Goal 1: Constructing a Dataset

The dataset is often not the flashiest part of a deep learning project, but as we constructed a relatively novel kind of dataset we will shortly discuss our conclusions and how they relate to the goals we defined.

Sentiment Extraction A lot of samples in our dataset are of dubious value, and that makes this problem challenging. While the Vader Sentiment extraction performed well, the visual sentiment extraction seemed more problematic. This is not unexpected: the dataset used in the MVSO project was compiled from Flickr images and their labels, it is likely that these images differ at least somewhat from ours, both visually and semantically. The clearest indications of suboptimal sentiment extraction performance are the very low correlations between features extracted from the images versus those from text, low correlations here is not necessarily bad, as it implies that the modalities carry different information. Them being zero however also implies that there is no relation between the sentiment found in tweets and included images, even when considering that the extractions target different values (visual targets secondary dyads [Plutchik, 1991], while textual targets different polarity measures), which intuitively seems unlikely.
Better results are likely possible with a dedicated model level multimodal approach. This would allow features from the text to be used while extracting visual sentiment, and vice versa. Training these models would however require a magnitude of data unavailable in this project, in addition to being a complex endeavour in and of itself. Initial experiments were done with straight up embedding images with the

convolutional parts of high performing pre-trained image classification models but these were problematic and again would require a bigger dataset.

6.1.2 Goal 2: Predicting Implied Volatility

As explained in the beginning of this text financial predictions are hard. While implied volatility is considered a relatively forgiving response variable, we found ourselves in a difficult predictive setting. While our models clearly showed predictive power, the results of our standard machine learning experiments, and our specialized experiments, both using solely multimodal data are not of the quality that would be expected a standalone prediction system. This part of the project therefore takes almost an ablative view, as we started with very simple models and approaches, of which we selected the best performing model. We finally demonstrated that our models can outperform the baseline of a lagged value when given access to a small amount of historical information, less than would likely be injected in practical applications.

Deep Learning Architectures and Inter-sample Interactions Deep Unordered Composition implemented by a CDAN is shown to be a competent and light-weight method for unordered sequence learning. Even when finely tuned, traditional models underperform our deep learning models. The neural architectures are also relatively small compared to what might be considered normal in deep learning. Even the model with the most trainable weights, a common measure of model complexity, has orders of magnitudes less trainable parameters than common models with similar input sizes[1]. The real stand-out feature is clearly the interaction between samples in the embedding stage, as used in the Pooling Nets. The multitude of paths for the information to flow and to be reconsidered allows our models to consider the whole sequence intelligently while embedding samples.

6.2 Possible Further Work

In deep learning there are two common ways to improve performance: training models on more data, and creating more complex models. The same applies to our models. In this section we will cover some possible areas of improvements, and some areas that might be less promising than initially thought.

6.2.1 Dataset

The most obvious available improvement is clearly the dataset, more data often means better performance, finding more data could mean taking another look at the scraping of the tweets, and see if our feeling of the dataset not being complete is justified. One could also gather a dataset on another company and then try to

[1] A fully connected model using the zero padding approach with one hidden layer of size 40 would have more parameters if the max input dimension would be 15x28, while our network fully agnostic to the sample size of the input.

look if models can be partially or fully shared. The feature extraction of our data is another area where improvements might seem possible, but the models used are considered to be some top performers in these applications:

Vader Vader is well known for its performance on social media text. Recent, more complex models for language tasks are often trained on corpuses with a seriously different kind of text. Recently however models pre-trained on similar text have been released [Dat Quoc Nguyen and Nguyen, 2020], but fine-tuning them to perform better specific to financial tweets would likely require a labelled dataset that is hard to acquire. Reworking the lexicon might be a good way forward but doing this well would again require data and resources.

MVSO Our choice of visual sentiment embedding is also a common one in related works, and serious improvements might be more challenging than expected. Fine-tuning an existing classifier, which is a common tactic in similar settings, did not really seem to give promising results. We consider it therefore unlikely that the embedding could be seriously improved, without previously increasing the dataset size significantly. Intense cleaning and preprocessing of the dataset is another option but it would inject a lot of specific knowledge into the system, obfuscating actual predictive value in a real setting.

6.2.2 More Complex Models

The models we use are simple, and it might seem that an area of improvement would be to just slap a more complex and/or more parametric model on the problem. This is likely harder than expected, even if a dataset of the necessary size could be compiled. The strongly varying input dimension coupled with the unordered nature of our sample make fully connected networks infeasible. Even sequence aware methods, such as RNN's, bidirectional LSTM's and autoregressive networks such as transformers[2] would likely not be able to learn these interactions, considering the small dataset featured in this project. A hybrid pyramid style embedding/aggregation approach with intermediate pooling layers aggregating samples is possible, but problems resulting from the strongly varying dimension will likely persist. Most other approaches will have to learn to deal with the structure and its properties, while our models are completely agnostic to it. For structurally better performance while only using the multimodal data one could also try to forego the extraction of sentiment and train a model end to end, going from a basic representation of text and images. Considering the model complexity likely required for this, in addition to the limited number of datapoints, it seems that a dataset orders of magnitude bigger would be necessary. Our concept of concentrating model complexity in the embedding part would likely be vital here. A more practical approach could be to get representations from pre-trained networks such as transformers for text and classifiers for the images.

[2] Our experiments with emulating attention and their lacklustre performance indicate that attention maybe is not all you need (in this task)

Optimally the networks should be pre-trained on data as similar as possible to the dataset.

6.3 Practicality

The models, even the best performing ones, are unlikely to be actually useful as a standalone model, as they fail to accurately predict IV. We however have to remember that we are using simple, light-weight models, using information from a small, noisy dataset to predict a complicated variable that is only partially dependent on the input data. The main drivers of the IV are large organizations employing a variety of competing highly complex models considering huge amounts of information. The fact that our model can outperform its dataset when provided with historical data is another indication that the architectures used could be very valuable. Especially when applied to the problem in larger models or ensemble settings. The goal of the project was to use only the multimodal data, and we tried to adhere to this principle. If the only objective was low errors and high accuracy the approach would be quite different. With thorough cleaning we could erode the dataset to only contain really meaningful tweets. It would however likely be hard to generalize this to a realistic real time setting, which would mean that our results would be less valuable. Another easy boost would be to inject fundamental information into the model post-aggregation. A production model for this task would likely use a lot more fundamental and market information. This was however not the scope of this research. The use of this information would, given the limited nature of our dataset, likely diminish the use of our multimodal features. It would also become difficult to evaluate the CDAN structures used as this information would likely be injected post aggregation.

6.4 Final Thoughts

In this project we created a unique dataset, which presented us a unique predictive setting. When considering the nature and size of this dataset, and the general difficulty of predicting future variables in this setting, we conclude that multimodal sentiment analysis of microblogs shows promise to predict future Implied Volatility, and the CDAN is an efficient strategy to learn the specific predictive setting, especially when modifications are made to allow for interaction between different tweets.

Appendices

Appendix A

Detailed Architecures of Deep Learning Models

In this appendix we'll show the layers used in our Deep Learning models.

Layer (type)	Input Shape	Param #
Dropout-1	[-1, 28, 50]	0
Pad_Pool_Res_Block-2	[-1, 28, 50]	0
PadPoolLayer-3	[-1, 28, 50]	0
MaxPool1d-4	[-1, 28, 60]	0
AvgPool1d-5	[-1, 28, 60]	0
Conv1d-6	[-1, 84, 50]	2,380
Conv1d-7	[-1, 28, 50]	812
Pad_Pool_Res_Block-8	[-1, 28, 50]	0
PadPoolLayer-9	[-1, 28, 50]	0
MaxPool1d-10	[-1, 28, 60]	0
AvgPool1d-11	[-1, 28, 60]	0
Conv1d-12	[-1, 84, 50]	2,380
Conv1d-13	[-1, 28, 50]	812
Pad_Pool_Res_Block-14	[-1, 28, 50]	0
PadPoolLayer-15	[-1, 28, 50]	0
MaxPool1d-16	[-1, 28, 60]	0
AvgPool1d-17	[-1, 28, 60]	0
Conv1d-18	[-1, 84, 50]	2,380
Conv1d-19	[-1, 28, 50]	812
Pad_Pool_Res_Block-20	[-1, 28, 50]	0
PadPoolLayer-21	[-1, 28, 50]	0
MaxPool1d-22	[-1, 28, 60]	0
AvgPool1d-23	[-1, 28, 60]	0
Conv1d-24	[-1, 84, 50]	2,380
Conv1d-25	[-1, 28, 50]	812
Pad_Pool_Res_Block-26	[-1, 28, 50]	0
PadPoolLayer-27	[-1, 28, 50]	0
MaxPool1d-28	[-1, 28, 60]	0
AvgPool1d-29	[-1, 28, 60]	0
Conv1d-30	[-1, 84, 50]	2,380
Conv1d-31	[-1, 28, 50]	812
CombinedAdaptivePool-32	[-1, 28, 50]	0
AdaptiveAvgPool1d-33	[-1, 28, 50]	0
AdaptiveMaxPool1d-34	[-1, 28, 50]	0
Linear-35	[-1]	570
Linear-36	[-1]	11
Total params	16,541	
Trainable params	16,541	
Non-trainable params	0	

Table A1: The Residual Pooling Network, with 5 residual blocks. All the convolutional layers are organized in blocks.

Layer (type)	Input Shape	Param #
Dropout-1	[-1, 28, 50]	0
Conv1d-2	[-1, 28, 50]	812
Conv1d-3	[-1, 28, 50]	812
Conv1d-4	[-1, 28, 50]	812
Conv1d-5	[-1, 28, 50]	812
Conv1d-6	[-1, 28, 50]	812
Conv1d-7	[-1, 28, 50]	812
Conv1d-8	[-1, 28, 50]	812
Conv1d-9	[-1, 28, 50]	812
Conv1d-10	[-1, 28, 50]	812
Conv1d-11	[-1, 28, 50]	812
AdaptiveAvgPool1d-12	[-1, 28, 50]	0
Total params	8,120	
Trainable params	8,120	
Non-trainable params	0	

TABLE A2: The Residual Pre Network, with 5 residual blocks. All the convolutional layers are organized in blocks.

Layer (type)	Input Shape	Param #
Dropout-1	[-1, 28, 50]	0
AttLayer-2	[-1, 28, 50]	0
Conv1d-3	[-1, 28, 50]	145
Conv1d-4	[-1, 33, 50]	680
AttLayer-5	[-1, 20, 50]	0
Conv1d-6	[-1, 20, 50]	105
Conv1d-7	[-1, 25, 50]	390
AttLayer-8	[-1, 15, 50]	0
Conv1d-9	[-1, 15, 50]	80
Conv1d-10	[-1, 20, 50]	315
AttLayer-11	[-1, 15, 50]	0
Conv1d-12	[-1, 15, 50]	80
Conv1d-13	[-1, 20, 50]	210
CombinedAdaptivePool-14	[-1, 10, 50]	0
AdaptiveAvgPool1d-15	[-1, 10, 50]	0
AdaptiveMaxPool1d-16	[-1, 10, 50]	0
Linear-17	[-1]	210
Linear-18	[-1]	11
Total params	2,226	
Trainable params	2,226	
Non-trainable params	0	

Table A3: Our attention inspired model Att Net, layers can attend to the global state of their input

Layer (type)	Input Shape	Param #
Conv1d-1	[-1, 28, 50]	290
AdaptiveAvgPool1d-2	[-1, 38, 50]	0
AdaptiveMaxPool1d-3	[-1, 38, 50]	0
Linear-4	[-1]	770
Linear-5	[-1]	11
Total params	1,071	
Trainable params	1,071	
Non-trainable params	0	

Table A4: Mixed net: One of our simplest networks, a pure implementation of a CDAN.

Layer (type)	Input Shape	Param #
Conv1d-1	[-1, 28, 50]	812
Conv1d-2	[-1, 28, 50]	812
Conv1d-3	[-1, 28, 50]	812
Conv1d-4	[-1, 28, 50]	812
Conv1d-5	[-1, 28, 50]	812
Conv1d-6	[-1, 28, 50]	812
Conv1d-7	[-1, 28, 50]	812
Conv1d-8	[-1, 28, 50]	812
Conv1d-9	[-1, 28, 50]	812
Conv1d-10	[-1, 28, 50]	812
AdaptiveAvgPool1d-11	[-1, 28, 50]	0
AdaptiveMaxPool1d-12	[-1, 28, 50]	0
Linear-13	[-1]	570
Linear-14	[-1]	11
Total params	8,701	
Trainable params	8,701	
Non-trainable params	0	

TABLE A5: A residual version of our Mixed Net network

Layer (type)	Input Shape	Param #
Conv1d-1	[-1, 4, 500]	10
PlusPoolLayer-2	[-1, 2, 500]	0
PadPoolLayer-3	[-1, 32, 50]	0
MaxPool1d-4	[-1, 32, 60]	0
AvgPool1d-5	[-1, 32, 60]	0
Conv1d-6	[-1, 96, 50]	1,940
PadPoolLayer-7	[-1, 20, 50]	0
MaxPool1d-8	[-1, 20, 60]	0
AvgPool1d-9	[-1, 20, 60]	0
Conv1d-10	[-1, 60, 50]	915
PadPoolLayer-11	[-1, 15, 50]	0
MaxPool1d-12	[-1, 15, 60]	0
AvgPool1d-13	[-1, 15, 60]	0
Conv1d-14	[-1, 45, 50]	460
PadPoolLayer-15	[-1, 10, 50]	0
MaxPool1d-16	[-1, 10, 60]	0
AvgPool1d-17	[-1, 10, 60]	0
Conv1d-18	[-1, 30, 50]	310
Conv1d-19	[-1, 4, 500]	15
CombinedAdaptivePool-20	[-1, 42, 50]	0
AdaptiveAvgPool1d-21	[-1, 42, 50]	0
AdaptiveMaxPool1d-22	[-1, 42, 50]	0
CombinedAdaptivePool-23	[-1, 3, 500]	0
AdaptiveAvgPool1d-24	[-1, 3, 500]	0
AdaptiveMaxPool1d-25	[-1, 3, 500]	0
Linear-26	[-1]	910
Linear-27	[-1]	11
Total params	4,571	
Trainable params	4,571	
Non-trainable params	0	

Table A6: Pooling Net Plus: This model leverages extra monomodal data by distributive pooling.

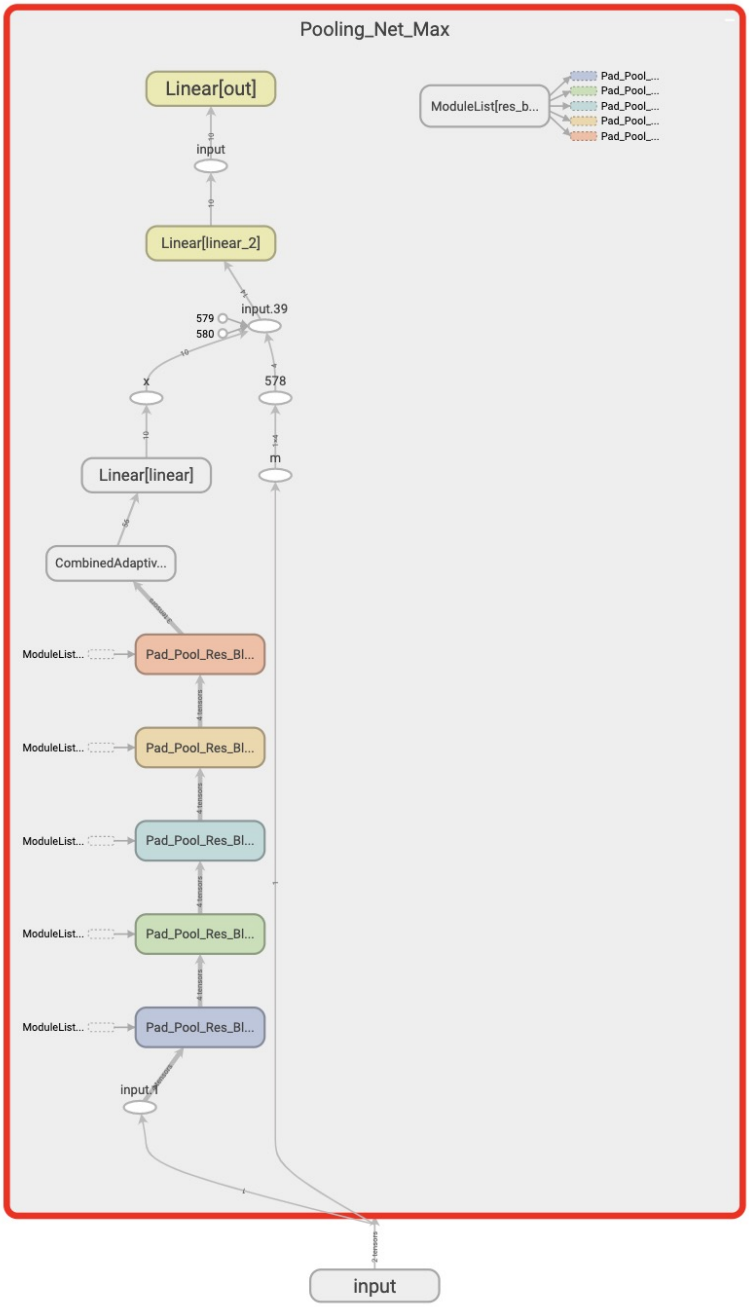


FIGURE A1: Pooling Net (residual) with IV history

Appendix B

Technical Context

All our experiments were performed in a certain technical context. In this section we explain the choices made.

B.0.1 Programming Environment

The code used and written all was located in Python (v3.8), which is a common environment for machine learning.

Python, Numpy and Multiprocessing

For data manipulation we mainly used Numpy. This package provides a lot of high-level functionality for arrays with multiple dimensions, mostly implemented in C meaning that it escapes the global interpreter lock and has more freedom in multiprocessing and vectorization. This results in much better performance compared to self-written code. A lot of multiprocessing was also used. This compares to multithreading in that since a different process is used the memory is not shared. This means we have to explicitly synchronize interprocess communication. It however allows us to leverage the many threads available. The use of multiprocessing in this project could be divided into two different approaches, one where we explicitly create different subprocesses with different functionality that interact. This was done in the scraping. Another one is where we create a pool of worker processes that we assign tasks, this was done in the preprocessing. Often we want different processes to work on a slab of shared memory, for this we'll use raw shared memory with implicit synchronization to avoid penalties resulting from locks and waiting.

A special case was the scraping engine, here we used a process pool an order of magnitude bigger than the amount of even logical cores available. This is highly uncommon but here it makes sense because the scraping task is highly IO constrained and this high process count will allow us to utilize more of the internet connection, which is still the bottleneck for this application.

Pytorch

PyTorch [Paszke et al., 2019] is the deep learning framework used in this project. We chose it because of its general performance, its python-like interface, and the freedom it allows us to manipulate the data, compared to some simpler frameworks like for example keras. Another good option was TensorFlow, but we judged that the more data-focused PyTorch, with its more mature dynamical graph was the better choice. The advantages of TensorFlow, such as more support for deployment, were deemed not to be highly relevant.
Writing PyTorch code looks a lot like normal python code, but there are a lot of things happening under the hood. Everything is based on the $torch.Tensor$ type. It is an internal representation of a multidimensional array. It is on these tensors that we will perform all operations during training and inference. Most functionality is implemented in C++ as is common with high-performance computing packages in Python(e.g. Numpy, TensorFlow). Tensors are however much more than just simple n-dimensional arrays, an important feature is automatic differentiation, PyTorch will store gradients generated by operations in certain conditions. This can be done with torch specific operations but also with normal python statements using overloading. This allows us to later perform a backwards pass to accumulate the gradients and perform a learning step. The way all this is functions efficiently is that data flow is split from control flow, largely handled by optimized C++ code.

B.0.2 Genius/Vsc

The first parts of the project were all run on either the computers of the computer science department or on collab, a free service that provides GPU's. Colab's drawbacks and performance limits (mostly ram, disk space, and CPU performance) pushed us to look for alternatives. This is where we arrived at the Vlaamse Supercomputer Center. This allowed us good performance, strong GPU compute, and distributed computing possibilities.

Bibliography

[Audrino et al., 2020] Audrino, F., Sigrist, F., and Ballinari, D. (2020). The impact of sentiment and attention measures on stock market volatility. *International Journal of Forecasting*, 36(2):334 – 357.

[Behrendt and Schmidt, 2018] Behrendt, S. and Schmidt, A. (2018). The twitter myth revisited: Intraday investor sentiment, twitter activity and individual-level stock return volatility. *Journal of Banking and Finance*, 96:355 – 367.

[Borth et al., 2013] Borth, D., Chen, T., Ji, R., and Chang, S.-F. (2013). Sentibank: large-scale ontology and classifiers for detecting sentiment and emotions in visual content. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 459–460.

[Breiman, 1997] Breiman, L. (1997). Arcing the edge.

[Cabanski et al., 2017] Cabanski, T., Romberg, J., and Conrad, S. (2017). HHU at SemEval-2017 task 5: Fine-grained sentiment analysis on financial data using machine learning methods. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 832–836, Vancouver, Canada. Association for Computational Linguistics.

[Cai and Xia, 2015] Cai, G. and Xia, B. (2015). Convolutional neural networks for multimedia sentiment analysis. In Li, J., Ji, H., Zhao, D., and Feng, Y., editors, *Natural Language Processing and Chinese Computing*, pages 159–167, Cham. Springer International Publishing.

[Campos et al., 2017] Campos, V., Jou, B., and Giro-i Nieto, X. (2017). From pixels to sentiment: Fine-tuning cnns for visual sentiment prediction. *Image and Vision Computing*, 65:15–22.

[Chen et al., 2014a] Chen, T., Borth, D., Darrell, T., and Chang, S.-F. (2014a). Deepsentibank: Visual sentiment concept classification with deep convolutional neural networks.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754.

[Chen et al., 2014b] Chen, T., Yu, F. X., Chen, J., Cui, Y., Chen, Y.-Y., and Chang, S.-F. (2014b). Object-based visual sentiment concept analysis and application. In *Proceedings of the 22nd ACM International Conference on Multimedia*, MM 14, pages 367–376, New York, NY, USA. Association for Computing Machinery.

[Cortis et al., 2017] Cortis, K., Freitas, A., Daudert, T., Huerlimann, M., Zarrouk, M., Handschuh, S., and Davis, B. (2017). Semeval-2017 task 5: Fine-grained sentiment analysis on financial microblogs and news. Association for Computational Linguistics (ACL).

[Dat Quoc Nguyen and Nguyen, 2020] Dat Quoc Nguyen, T. V. and Nguyen, A. T. (2020). BERTweet: A pre-trained language model for English Tweets. *arXiv preprint*, arXiv:2005.10200.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[Edwards and Storkey, 2016] Edwards, H. and Storkey, A. (2016). Towards a neural statistician. *arXiv preprint arXiv:1606.02185*.

[Exchange, 2009] Exchange, C. B. O. (2009). The cboe volatility index-vix. *White Paper*, pages 1–23.

[Friedman, 2001] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232.

[Fukushima, 1988] Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119 – 130.

[Gardner et al., 2019] Gardner, A., Elhami, N., and Selmic, R. R. (2019). Classifying unordered feature sets with convolutional deep averaging networks. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 3447–3453.

[Glodek et al., 2013] Glodek, M., Reuter, S., Schels, M., Dietmayer, K., and Schwenker, F. (2013). Kalman filter based classifier fusion for affective state recognition. In Zhou, Z.-H., Roli, F., and Kittler, J., editors, *Multiple Classifier Systems*, pages 85–94, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hochreiter et al., 2001] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[Howard and Ruder, 2018] Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

[Hull, 2018] Hull, J. (2018). *Options, Futures, and Other Derivatives 10th Edition*. Pearson.

[Hung and Chen, 2016] Hung, C. and Chen, S.-J. (2016). Word sense disambiguation based sentiment lexicons for sentiment classification. *Knowledge-Based Systems*, 110:224–232.

[Hutto and Gilbert, 2014] Hutto, C. J. and Gilbert, E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*.

[Iyyer et al., 2015] Iyyer, M., Manjunatha, V., Boyd-Graber, J., and Daumé III, H. (2015). Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1681–1691, Beijing, China. Association for Computational Linguistics.

[Jia et al., 2009] Jia, L., Yu, C., and Meng, W. (2009). The effect of negation on sentiment analysis and retrieval effectiveness. pages 1827–1830.

[Jou et al., 2015] Jou, B., Chen, T., Pappas, N., Redi, M., Topkara, M., and Chang, S.-F. (2015). Visual affect around the world: A large-scale multilingual visual sentiment ontology. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 159–168.

[Karpathy, 2020] Karpathy, A. (2020). Ai for full-self driving. *5th Annual Scaled Machine Learning Conference*.

[Kumar and Garg, 2019] Kumar, A. and Garg, G. (2019). Sentiment analysis of multimodal twitter data. *Multimedia Tools and Applications*, 78.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[Malkiel, 2003a] Malkiel, B. G. (2003a). The efficient market hypothesis and its critics. *Journal of economic perspectives*, 17(1):59–82.

[Malkiel, 2003b] Malkiel, B. G. (2003b). Passive investment strategies and efficient markets. *European Financial Management*, 9(1):1–10.

[Mansar et al., 2017] Mansar, Y., Gatti, L., Ferradans, S., Guerini, M., and Staiano, J. (2017). Fortia-fbk at semeval-2017 task 5: Bullish or bearish? inferring sentiment towards brands from financial news headlines. *arXiv preprint arXiv:1704.00939*.

[Mishne et al., 2005] Mishne, G. et al. (2005). Experiments with mood classification in blog posts. In *Proceedings of ACM SIGIR 2005 workshop on stylistic analysis of text for information access*, volume 19, pages 321–327.

[Oliveira et al., 2017] Oliveira, N., Cortez, P., and Areal, N. (2017). The impact of microblogging data for stock market prediction: Using twitter to predict returns, volatility, trading volume and survey sentiment indices. *Expert Systems with Applications*, 73:125 – 144.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., deBuc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Plutchik, 1991] Plutchik, R. (1991). *The Emotions*. G - Reference, Information and Interdisciplinary Subjects Series. University Press of America.

[Poria et al., 2017a] Poria, S., Cambria, E., Bajpai, R., and Hussain, A. (2017a). A review of affective computing: From unimodal analysis to multimodal fusion. *Information Fusion*, 37.

[Poria et al., 2015] Poria, S., Cambria, E., Hussain, A., and Huang, G.-B. (2015). Towards an intelligent framework for multimodal affective data analysis. *Neural Networks*, 63:104 – 116.

[Poria et al., 2017b] Poria, S., Peng, H., Hussain, A., Howard, N., and Cambria, E. (2017b). Ensemble application of convolutional neural networks and multiple kernel learning for multimodal sentiment analysis. *Neurocomputing*, 261:217–230.

[Qi et al., 2017] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660.

[Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

[Raffel et al., 2019] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*.

[Rashmi and Gilad-Bachrach, 2015] Rashmi, K. V. and Gilad-Bachrach, R. (2015). Dart: Dropouts meet multiple additive regression trees. In *AISTATS*, pages 489–497.

[Rentoumi et al., 2009] Rentoumi, V., Giannakopoulos, G., Karkaletsis, V., and Vouros, G. A. (2009). Sentiment analysis of figurative language using a word sense disambiguation approach. In *Proceedings of the International Conference RANLP-2009*, pages 370–375.

[Richard and Gall, 2017] Richard, A. and Gall, J. (2017). A bag-of-words equivalent recurrent neural network for action recognition. *Computer Vision and Image Understanding*, 156:79–91.

[Sardelich and Manandhar, 2018] Sardelich, M. and Manandhar, S. (2018). Multimodal deep learning for short-term stock volatility prediction. *arXiv preprint arXiv:1812.10479*.

[Savva et al., 2011] Savva, M., Kong, N., Chhajta, A., Fei-Fei, L., Agrawala, M., and Heer, J. (2011). Revision: Automated classification, analysis and redesign of chart images. In *ACM User Interface Software & Technology (UIST)*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[Wang et al., 2016] Wang, J., Fu, J., Xu, Y., and Mei, T. (2016). Beyond object recognition: Visual sentiment analysis with deep coupled adjective and noun neural networks. In *IJCAI*, pages 3484–3490.

Fiche masterproef

Student: Arthur Decloedt

Titel: Multimodal Tweet Sentiment for Implied Volatility Predictions

Nederlandse titel: Multimodaal tweet sentiment voor het voorspellen van geïmpliceerde volatiliteit

UDC: 621.3

Korte inhoud:
This thesis researches the extraction of multimodal sentiment from tweets referencing an equity and approaches on using it for predicting future implied volatility on that equity. A scalable and efficient data collection procedure is defined and implemented. The constructed dataset contains both textual and visual information with a sub day level granularity. Emotional features from both modalities are extracted using pre-trained models with a decision level fusion approach, resulting in a dataset reflecting popular sentiment towards a specific equity over multiple years with sentiment vectors for each tweet. We present several lightweight, size agnostic architectures, derived from convolutional deep averaging networks, that outperform out of the box machine learning approaches. While the used dataset is relatively small and rough, the neural networks show promise in this task. We finally demonstrate the models outperforming baselines set by the lagged value when relaxing the constraint of only using the multimodal data.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotor: Prof. dr. Jesse Davis, Prof. dr. Wim Schoutens

Assessor: ir. Tom Decroos, dr. Nyi-Nyi Htun

Begeleider: ir. Thomas Dierckx