Reflectie opdracht MOI: Aplied Data Science – Machine Learning

Doel van de opdracht

Deze opdracht heeft als doel te experimenteren met de FashionMNIST dataset en vervolgens op deze experimenten te reflecteren. In dit document wordt als eerste het gekozen model besproken, vervolgens worden de dataset en dataloader kort benoemd besproken. Het derde deel gaat in op het experimenteren met het gekozen model, de problemen die hier bij naar boven kwamen en de stappen die genomen zijn om deze problemen weer op te lossen. Als laatste een reflectie op de algehele ontwikkeling en de daarbij behorende bevindingen.

1. Het gekozen model

Tijdens de les van 2022-05-10 werd in het kort SENet besproken. Om mijzelf uit te dagen wilde ik dit model na bouwen in pytorch. Na enig zoeken kwam ik meerdere implementaties tegen van het SEnet. Specifiek ben ik mij gaan richten op de Squeezenet 1.1 architectuur. Het hart van dit model is de “fire module” De implementatie van deze architectuur wordt in deel 3 verder besproken.

2. De data en dataloader

De data betreft de FashionMNIST dataset van 1x28x28 afbeeldingen. Deze afbeeldingen bevatten zwart-wit foto’s van kleding artikelen. Het doel met deze dataset is om de afbeeldingen in de juiste categorie (10) te classificeren. Deze dataset wordt regelmatig gebruikt als baseline dataset voor neurale netwerken en een accuracy van >90% is momenteel de standaard op deze dataset. De verdeling classes over de images (train en test) is evenredig. Dit maakt het makkelijker om een model dat specifiek voor deze dataset wordt gebouwd te beoordelen.

De in les 1 aangeboden dataloader is gebruikt voor dit onderzoek. Er zijn geen aanpassingen of transformaties gedaan bij de input afbeeldingen. De dataloader is ingesteld op batches van 64 afbeelding. Dit gaf het beste resultaat in de baseline models en leek daarom het beste om te gebruiken als vergelijkingsmateriaal voor het testen van een nieuwe architectuur.

Omdat in deze opdracht geprobeerd wordt een nieuwe architectuur te implementeren die ik nog niet kende zijn er verder geen aanpassingen gedaan aan de dataloader. Dit zorgt voor de meest zuivere testen van de architectuur zelf.

3. Model architectuur en experimenten

De gekozen model architectuur betreft squeezenet 1.1 (zie link). is oorspronkelijk ontwikkelt op RGB afbeeldingen van met dimensies 3x227x227 (CxHxW). Het was dus vanaf het begin al duidelijk dat dit model “overkill” is voor afbeeldingen van 1x28x28. Maar goed voor dat dit netwerk geïmplementeerd kon worden moesten eerst de losse onderdelen geïmplementeerd worden. De eerste convolutie en pool laag vormde hier nog geen probleem. Dit hebben we tijdens de lessen geleerd en was makkelijk werkbaar te krijgen. De uitdaging kwam bij de implementatie van de fire module.

De eerste uitdaging

Hoe krijg ik het voor elkaar om een module die zelf meerdere bewerkingen uitvoert te implementeren in het model? Het eerste wat ik probeerde was de individuele onderdelen (squeeze, excite, concat) klaar te zetten in de *init* fase van het model en deze in de forward functie 1 voor 1 aan te roepen. Dit werkte maar leverde een brei aan code, onverwachte resultaten, lange train

tijden en problemen op met het leren van signalen. Na meer research online te hebben gedaan bleek dat de conventie is om dergelijke “modules” in een eigen klasse te plaatsen en deze mee te nemen in de uiteindelijk klasse van het model. Dit leek een deel van de problematiek op te lossen. Het model kon sneller trainen (van +/-2 minuten naar +/- 1 minuten per epoch). Echter leken de gewichten zich nu (nog steeds) niet aan te passen.

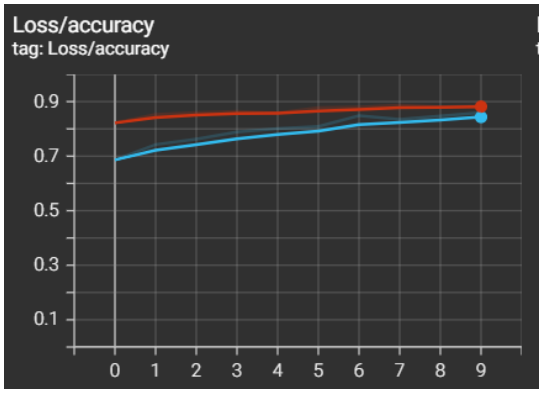
De tweede uitdaging

Nu het model draaide op een versimpelde versie van het netwerk bleken de gewichten zich niet te updaten. Het eerste vermoeden was dat het model geïnitialiseerd was in een lokaal minimum en hier niet uit ontsnapte. Maar met het aanpassen van de learning rate, meerdere keren het model te trainen en meer parameters toe te voegen bleek dit niet aan de orde te kunnen zijn. Het model had een accuracy van <10% en ontwikkelde zich niet. Na het probleem in de les besproken te hebben ben ik nogmaals gaan testen door het model opnieuw laag voor laag op te bouwen. Het bleek mis te gaan bij de fire module ik wist dus waar ik moest kijken. Met deze kennis ben ik de documentatie in pytorch gaan raadplegen. Het probleem bleek voor te komen uit de activatie functie. Deze *ReLu()* activatie moet gebruik maken van het argument *inplace=True*. De reden heb ik nog niet helemaal kunnen achterhalen, ook diverse fora (waaronder die van pytorch zelf) waar deze discussies plaats vonden waren niet duidelijk over dit probleem. Uiteindelijk is mijn vermoeden dat in dit geval de activatie functie de uitkomst niet correct wegschrijft waardoor de oude input tensor in memory bewaard blijft. Dit zou dan zorgen voor lineaire lagen zonder activatie en dus een model dat alleen lineaire berekeningen maakt waarbij de resultaten niet beter scoren dan gokken. *Inplace=True* overschrijft de variabele in memory waardoor zeker is dat de output van de activatie functie gebruikt wordt in het model.

4. reflectie

In dit relatief korte stuk heb ik getracht mijn experimenten en mijn gedachten te noteren om inzichten te bieden in mijn ontwikkeling op het gebied van machine learning. Ik leg de lat vaak hoog en denk dat ik dit keer niet anders heb gedaan. Soms voelde ik mij tijdens deze opdracht als een slechte programmeur / data scientist. Gelukkig heb ik achterhaald waar de fouten lagen en door deze uitdagingen veel meer inzichten kunnen krijgen in de werkingen van dergelijke modellen. Het belangrijkste wat ik nu meeneem is het compartimenteren van de code (modules en modellen in verschillende klassen verwerken). Ook de noodzaak om de data goed te begrijpen en het model hierop aan te passen is mij duidelijk geworden. Voornamelijk het feit dat meer lagen en parameters doorgaans niet de oplossing is. In dit geval zou het conceptuele squeezenet-fire model gebouwd voor 3x227x227 afbeeldingen mogelijk leiden tot overfitting en in dit specifieke geval niet eens werken doordat de afbeelding te klein is om zo vaak gecomprimeerd te worden.

Uiteindelijk heb ik een SEFireNet kunnen maken met 4.296.103 parameters en deze vergelijken met een basis convolutioneel netwerk met 3.706.859 parameters (het SEFireNet is hiermee +/- 15% groter).



De resultaten rechts laat de accuracy van zowel SEFireNet (blauw) als het basis netwerk (rood) zien. Het doel was niet om de hoogste accuracy te halen maar een nieuw netwerk te bouwen en dat is gelukt! Toch ben ik benieuwd of dit nieuwe netwerk uiteindelijk het basis netwerk kan verslaan wanneer dit langer dan 10 epochs trained. En gezien de afvlakking van de rode lijn lijkt dit mogelijk.