Introductie

slide1:
Welkom iedereen bij de literatuurstudie voor het onderwerp: ‘Sensitivity analysis and uncertainty quantification of deep learning methods.’.

Deep learning en NN

slide2:
Zoals de titel al doet vermoeden is een belangrijke component van dit onderwerp deep learning. Indien we deep learning binnen het geheel van artificiële intelligentie bekijken zien we dat dat eigenlijk een sub-genre is van machine learning. Deep learning bevat ook wel veel bekende buzzwords uit da AI zoals ‘neural networks’ of ‘image classification’. Image classification is dan ook wel 1 van de toepassingen van deep learning, maar ook spraakherkenning is een voorbeeld waarvoor deep learning goed kan dienen.

slide3:
Zoals eerder gezegd is deep learning geassocieerd met neurale netwerken en om iets van deep learning te kunnen begrijpen is het belangrijk dat je ook begrijpt hoe deze netwerken juist werken.

In een neuraal netwerk kunnen we de verschillende lagen opdelen in 3 categorieën zoals ook te zien is op de foto. We hebben de inputlaag, de verborgen lagen en de outputlaag. We spreken ook wel over lagen van neurons waarbij elke cirkel een neuron voorstelt.

Het grootste deel van het werk gebeurt in de verborgen lagen. Elk van deze lagen voert een lineaire transformatie uit op de data die de laag als input krijgt, gevolgd door een niet lineaire transformatie en het resultaat hiervan wordt aan de volgende laag doorgegeven.

Voor de niet lineaire transformatie gaan we de input vermenigvuldigen met een gewichtsmatrix en daarbij een bias of vooroordelen vector bij op tellen. Een voorbeeld van een niet lineaire transformatie is de sigmoid functie waar het resultaat van de lineaire transformatie aan meegegen wordt.

Nadat de input door al de verborgen lagen is gestuurd en al die transformaties is ondergaan genereert het netwerkt een output aan de outputlaag.

slide4:
Indien we dit concreet voorstellen met een input vector x, gewichtsmatrix w, bias vector b en niet lineaire transformatie f kunnen we de eerste lineaire transformatie voorstellen zoals op de foto te zien is.

Elke neuron geeft dus een gewicht aan elk van de input elementen om de gewichtsmatrix te vormen. Hier wordt de bias vector bij opgeteld die de zogenaamde vooroordelen van elke neuron bevat. Dit geeft ons een nieuwe vector die, na het uitvoeren van de niet lineaire transformatie op die vector, dient als input voor de volgende laag neurons waar we opnieuw

een lineaire transformatie hebben, gevolgd door een niet lineaire en het hele proces zich herhaald.

Wanneer we input sturen door het netwerk is het resultaat dus een opeenvolging van lineaire en niet lineaire transformaties. Voor iedere laag kunnen we dan de uitgevoerde bewerkingen tot aan die laag uitschrijven zoals op de volgende slide.

(animatie verder gaan tot slide met pijlen)

We beginnen dus aan de input laag met de vector x die doorgegeven wordt aan elk van de neurons uit de volgende laag. Die laag voert een eerste lineaire en niet lineaire transformatie uit waarvan het resultaat naar de volgende laag gaat en we duidelijk de opeenvolging van transformaties zien. Dit gaat verder tot we uiteindelijk aan de laatste laag komen die de output van het netwerk berekend.

Netwerk leren

slide 5:
Hoe kunnen we nu het voorgaande gebruiken om het netwerk concepten aan te leren? We hebben nood aan data waarvan we de output op voorhand kennen. Vervolgens initialiseren we de gewichten van het netwerk op een willekeurige manier en we berekenen de accuraatheid van de voorspellingen op onze data. Door de willekeurigheid is ze bij de initialisatie nooit erg goed. Deze accuraatheid is enkel afhankelijk van de gewichten en de biases van het model en deze zullen dus betere waarden moeten verkrijgen. Dit doen we door het optimaliseren van de parameters met behulp van achterwaartse propagatie.

Het algoritme dat hiervoor kan gebruikt worden is gradient descent. Hierbij berekenen we het gradient van de kostfunctie van alle punten van de dataset. Deze functie evalueert de accuraatheid van de trainingspunten. Met deze gradiënt gaan we de kant kiezen naar waar we de parameters gaan shiften om een betere prestatie te verkrijgen.

Je kan op de tekening op de slide zien dat we vertrekken van een gebied met een hoge kost en zo stap voor stap dalen naar een kleinere kost met behulp van de gradiënt. Het algoritme dat op deze tekening gebruikt wordt is wel de stochastische versie van GD, die we verder gaan bespreken.

Metaparameters

slide 6:
Indien we de update regel uit de vorige slide wat nader bekijken zien we dat er 2 delen zijn die we moeten bepalen. Namelijk de stap-grootte en de gradiënt van de kostfunctie. De stap-grootte is vrij te kiezen, maar de keuze tussen een kleine of grote stap heeft wel gevolgen voor de optimalisatie van de kostfunctie. Met een kleine stap zal het leren iets trager verlopen, maar kan de optimalisatie wel fijner verlopen dan wanneer we een grotere stap nemen. De figuur op de slide toont een probleem dat kan optreden voor een grote stap waarbij we de optimalisatiestap wel in de correcte richting gaan zetten, maar door de te grote stap altijd het minimum gaan voorbijschieten en niet veel meer optimaliseren.

Voor het berekenen van de gradiënt is er ander probleem dat zal optreden.
slide7:
We zouden de kost kunnen voorstellen aan de hand van de voorbeeld kostfunctie op de slide.
Waarbij we N datapunten xi hebben waarvoor ons deep learning model als resultaat aL(xi) geeft en we ook het bijbehorende correcte resultaat yi kennen. De kostfunctie stelt in dit geval een soort van gemiddelde afstand voor tussen het resultaat van ons model en het correcte resultaat.

Verder kan er ingezien worden dat we elk van de termen in de sommatie van de kostfunctie als een functie opzich kunnen beschouwen. Hierdoor kan het berekenen van de gradiënt gezien worden als het berekenen van de sommatie van de gradiënten van de afzonderlijke deelfuncties.

Om dan de volledige gradiënt te bepalen is het aantal te berekenen afzonderlijke gradiënten gelijk aan het aantal datapunten.

slide8:

Dit is praktisch niet haalbaar omdat een model vaak veel lagen en neurons heeft, wat ervoor zorgt dat we veel parameters hebben voor de kostfunctie en omdat een model ook meestal zeer veel datapunten gebruikt.

In het geval dat we al maar 10000 datapunten hebben zal het berekenen van deze gradiënt een stevige operatie zijn.

Hiervoor moet dus een oplossing bestaan.

slide9:

De oplossing bestaat er uit om niet alle datapunten mee in rekening te gaan brengen voor het berekenen van de gradiënt voor de update regel.

Een eerste optie is om telkens maar 1 datapunt in rekening te brengen en dus enkel de gradiënt die hoort bij de deelfunctie van dat datapunt te gaan berekenen. De update-regel ziet er in dat geval dan uit zoals de eerste formule op slide.

Een 2de optie is om niet 1, maar een willekeurig gekozen aantal datapunten ‘m’ mee in rekening te brengen waarbij m veel kleiner is dan het totaal aantal datapunten N. De verzameling van de datapunten die in rekening worden gebracht noemt een zogenaamde ‘mini-batch’. De grootte van de mini-batch is een parameter die dan ook moet bepaald worden. De update regel ziet er in dit geval uit zoals de 2de formule op de slide. Hiervoor moet dan logischer wijze de gradiënt worden berekend van de deelfuncties die horen bij elk van de datapunten uit de mini-batch.

Deze 2 methodes kunnen zowel met als zonder vervanging. Hiermee wordt aangegeven of datapunten die al reeds mee in rekening gebracht zijn voor het updaten van de parameter p

opnieuw in rekening gebracht mogen worden alvorens alle andere datapunten in rekening gebracht zijn.

slide 10:
De training eindigt nadat er ofwel een vooraf bepaalde tijdslimiet bereikt is of de gradient heel klein wordt(+/-0). We hebben nu hopelijk een goed geoptimaliseerde kostfunctie. We kunnen nu het netwerk testen op een subset van data die we op voorhand voorgehouden hebben om te testen. Als de performantie goed is dan kunnen we het netwerk gebruiken voor bijvoorbeeld classificatie. Anders heeft het model toch geen goede parameters en gaan we de meta parameters aanpassen en het algoritme opnieuw laten beginnen.

Zekerheid netwerk

slide 11:
De zekerheid van klassieke neurale netwerken is moeilijk vast te stellen. Ze zijn namelijk vaak te zelfzeker van hun voorspellingen en geven dus geen duidelijke maat voor hun onzekerheid. Voor bepaalde toepassingen zoals zelfrijdende autos of autonome reactoren zou het belangrijk moeten zijn, als we niet heel zeker zijn van onze voorspelling, om toch te reageren met een actie die voor minder risico zorgt dan degene die het netwerk als de voornaamst koos. Bijvoorbeeld als een zelfrijdende auto slechts 60% zeker is dat een weg niet afgesloten is ze eerder gaat vertragen dan verder rijden op de voorgaande snelheid. Hiervoor volstaan klassieke NN niet en zoeken we naar een nieuwe oplossing.

BNN

slide 12:
We introduceren een nieuw concept: Bayesiaanse neurale netwerken. Deze verschillen van klassieke netwerken dat ze geen concrete gewichten hebben maar verdelingen van gewichten.

Stabiliteit

(overgang nog wa onduidelijk omda ik het voorgaande nog ni heb)
slide 16:
We gaan nu kijken naar de stabiliteit van SGD. We definiëren eerst de stabiliteit. Het algoritme is epsilon-stabiel als we trainen met 2 datasets die verschillen in juist 1 datapunt en de uitkomst begrensd is door epsilon.