Eindopdracht Computer Vision

Door Yoey Tolboom - 1770257

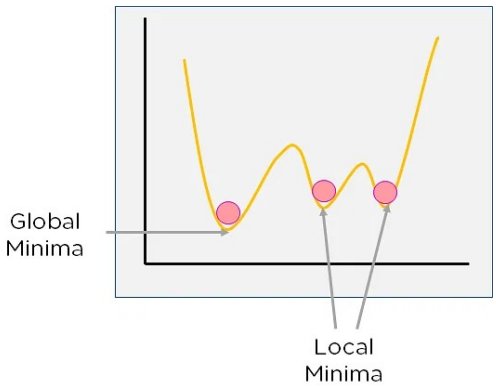
Introductie

Neurale netwerken zijn breed toepasbaar voor computer vision taken. Het zijn in feite functie approximaties; de mogelijkheden zijn eindeloos. Dit komt echter niet zomaar; neurale netwerken hebben duizenden individuele trainingen nodig om nuttig te worden. Dit vereist niet alleen een grote dataset, maar het kan ook enorm lang duren, vooral wanneer elk individueel stuk input data groot is. Het is daarom van groot belang dat neurale netwerken zo goed mogelijk presteren in een zo kort mogelijke tijd.

Achtergrond

Forward propagation
Forward propagation begint bij de inputs van het neurale netwerk en werkt naar de outputs. De waarden van de neuronen in de volgende laag worden berekend door alle weights te vermenigvuldigen met de waarden van de neuronen in de vorige laag, er daarna een bias bij op te tellen en die waarde vervolgens in een functie te doen die het domein tussen 0.0 en 0.1 houdt. 2 functies die dit bijvoorbeeld doen zijn de Sigmoid en de RELU.

Cost functies en Gradient descent
De cost functie geeft aan hoe slecht de output was; hoe hoger de cost, hoe slechter de output. Deze functie heeft dus alle weights en biases van het neurale netwerk als input (omdat de outputs van het neurale netwerk afhankelijk zijn van alle weights en biases) en geeft gebaseerd daarop 1 getal. Het doel hier is om de output van de cost functie zo laag mogelijk te krijgen. Het proces om dit te kunnen bereiken heet gradiënt descent. Het lastige aan gradiënt descent is dat de cost functie vaak duizenden inputs heeft. Elk van deze inputs kan omhoog of omlaag en zou daarmee de output kunnen veranderen. Daarom is het makkelijker om het probleem terug te schalen naar minder dimensies. Zie afbeelding 1, de gele lijn is in dit voorbeeld de Cost functie. Hieruit is te zien dat er een aantal dallen zijn waar de Cost functie laag is. Het laagste punt van deze dallen heet het globale minimum en de overige laagste punten in dallen heten lokale minima. Net als de rode cirkels in het simpele voorbeeld is het doel dus om één van deze minima te vinden, maar dan in een n-dimensionele ruimte, waar n het aantal weights en biases is. Een van de algoritme die dit doet heet backward propagation, dit algoritme is ook gebruikt om het neurale netwerk te implementeren.



Afbeelding 1 (Banoula, 2023)

Nudge factor
Voor dit experiment is er een extra parameter toegevoegd die getest gaat worden: de nudge factor. Deze bepaald hoe snel het neurale netwerk beweegt naar dat lokale minimum in de gradient descent functie.

Methoden

Om te meten hoe goed een neuraal netwerk presteert, wordt de gemiddelde loss berekent van alle plaatjes in de testing set. Deze staat bij elke bijbehorende afbeelding van een test op de y-as. De methode die gebruikt wordt om loss te berekenen is Mean Squared Error (MSE). Dat wordt berekent als volgt:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2.$$

Y is het geobserveerde antwoord.
Ŷ is het optimale antwoord; dit is dus 1.0 als die bepaalde output de juiste was en anders 0.0.
(Y – Ŷ)^2 is dan de ‘error’, en de MSE is dus het gemiddelde van de som van alle errors.

Dataset
Voor alle tests wordt de mnist dataset (MNIST as .Jpg) gebruikt. Deze bevat 70.000 plaatjes (28x28 pixels) van handgeschreven cijfers, waarvan er 42.000 gebruikt zijn voor het trainen en testen. Dit is een erg standaard computer vision dataset voor het testen van neurale netwerken en daarom wel gepast voor efficiëntie test. Alle afbeeldingen uit deze dataset die in het neurale netwerk worden gevoed zijn grayscaled. Alle waarden in de input layer kunnen dan simpelweg tussen de 0.0 (zwart) en 1.0 (wit) zijn.

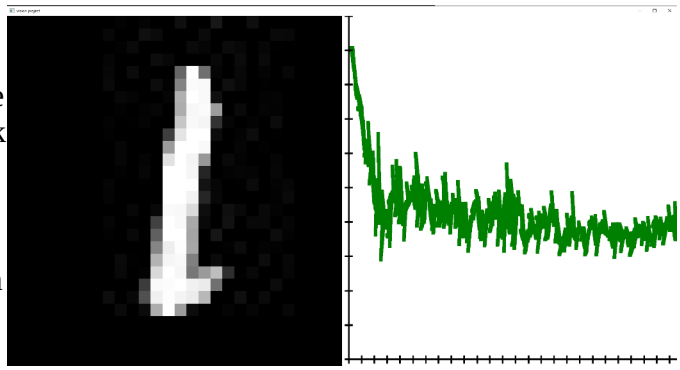
Structuur
Het neurale netwerk heeft altijd een input laag met 1 neuron per pixel. Als de dataset 28x28 pixels heeft per afbeelding, dan heeft de input laag 784 neuronen. De output laag heeft altijd hetzelfde aantal neuronen als het aantal mogelijke antwoorden. Voor de mnist data set is dit dus 10, omdat er 10 verschillende cijfers zijn die het neurale netwerk moet kunnen herkennen. Tussen de input laag en de output laag zit nog een variërend aantal hidden layers, elk met een variërend aantal neuronen. De logica achter hidden layers is dat ze bepaalde patronen in de afbeeldingen zouden kunnen herkennen en dit door kunnen geven naar de volgende layer.

Code
De code waar alle experiment opzet in geschreven is, is C. De rendering wordt allemaal gedaan met OpenGL. Daarvoor was glew (GLEW: The OpenGL Extension Wrangler Library) nodig om de functies te kunnen gebruiken. Voor de windowing system en input is glfw (An OpenGL Library) gebruikt. Om afbeeldingen in te laden is stb_image (nothings/stb) gebruikt, deze is inbegrepen in de code als een header file.

De training en testing omgeving is gemaakt in een statemachine die door de volgende states kan gaan: Start, Training, Testing, Quit. In Start wordt alles geinitialiseerd en worden de parameters voor het neurale netwerk aangemaakt (zie afbeelding 2). In Training en Testing wordt het neurale netwerk met die parameters getraind en getest. En in de Quit state wordt de applicatie klaar gemaakt om af te sluiten. Voor automatisatie is het heel makkelijk om de code om te zetten naar een loop (Training → Testing en weer terug) waar de parameters steeds iets veranderen gebaseerd op het experiment. Om elk experiment af te sluiten wordt er ook nog een grafiek gegenereerd aan het einde (zie afbeelding 3)

```
TrainingArgs* tArgs = (TrainingArgs*)malloc(sizeof(TrainingArgs));
tArgs->neuronsPerLayer[0] = 50;
tArgs->neuronsPerLayer[1] = 1;
tArgs->LayerCount = 2;
tArgs->nudgeFactor = 0.1;
s->data1 = (void*)tArgs;
```

Afbeelding 2



Afbeelding 3

Random
Voor elk van de experimenten is dezelfde seed gebruikt om willekeurige getallen te genereren. Dit is zo gedaan om te voorkomen dat sommige neurale netwerken toevallig een betere training volgorde krijgen. Of dat sommige weights en biases toevallig beter of slechter stonden in bepaalde neurale netwerken. Dit is nodig om de resultaten zo deterministisch en eerlijk mogelijk te houden.

Parameters
Elk experiment zal de invloed van 1 variabele tegelijk testen, alle andere variabelen zullen constant blijven. Daarnaast zal elke test iteratief proberen te verbeteren op basis van de vorige testen. Alle neurale netwerken met dezelfde dataset zullen hetzelfde aantal afbeeldingen krijgen om te trainen en zullen hetzelfde aantal testafbeeldingen beoordelen.

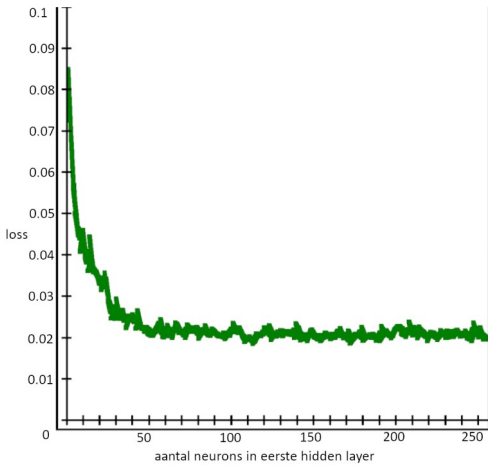
Experimenten

De volgende variabelen zullen op volgorde getest worden:

- aantal neuronen in de eerste hidden layer
- nudge factor
- aantal neuronen in de tweede hidden layer

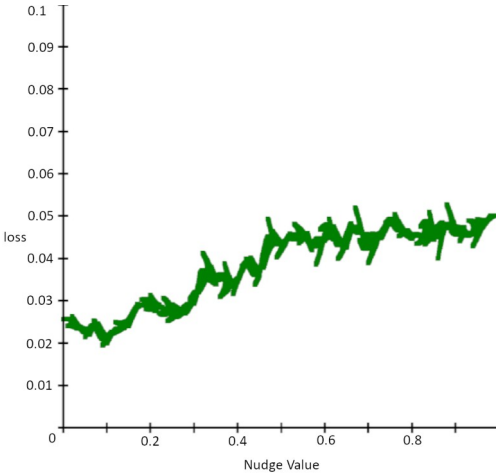
Wat al deze variabelen gemeen hebben is dat ze ongetwijfeld invloed hebben op de prestatie van elk neurale netwerk. Daarom worden deze getest.

Neuronen in de eerste hidden layer (afbeelding 4)
Voor deze test zijn 1-256 neuronen in de eerste layer getest. Elk neurale netwerk heeft 1 hidden layer en een nudge factor van 0.12 (toen nog arbitrair gekozen). Op de x-as staat het aantal neuronen en op de y-as staat de bijbehorende loss, oftewel de gemiddelde kwaliteit van het antwoord over 42000 test afbeeldingen. Na circa 50 neuronen op de eerste hidden layer hebben meer neuronen veel minder invloed op de loss functie. Dit zal dus de constante zijn in de volgende test.



Afbeelding 4

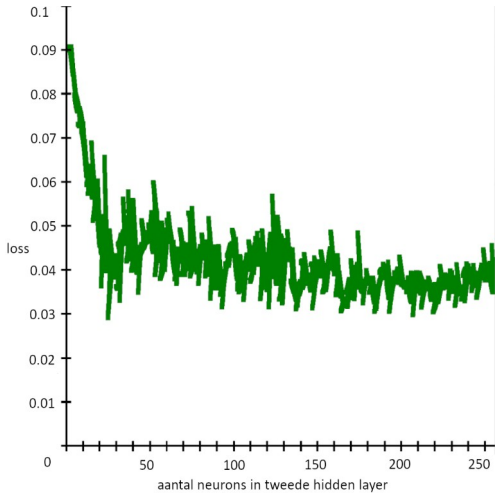
Nudge factor (afbeelding 5)
Voor deze test is elke nudge factor tussen de 0.01 en 1.0 getest met een interval van 0.01, 100 samples dus. Het neurale netwerk heeft 1 hidden layer met 50 neuronen. Uit afbeelding blah is duidelijk dat een lagere nudge factor rond de 0.1 het beste resultaat geeft met de gegeven parameters. Dit zal de constante worden voor de volgende test.



Afbeelding 5

Neuronen in de tweede hidden layer (afbeelding 6)
Voor deze test zijn 1-256 neuronen in de tweede layer getest. Elk neurale netwerk heeft in de eerste layer 50 neuronen en heeft een nudge factor van 0.1. Op de x-as staat het aantal neuronen en op de y-as staat de bijbehorende loss.

Het beste neurale netwerk
Het beste neurale netwerk in dit experiment heeft 1 hidden layer met 50 neuronen en een nudge factor van 0.1. Een gemiddelde loss van ongeveer 0.018, en had ongeveer 89% van alle test afbeeldingen goed geraden. Het Trainen van dit neurale netwerk duurde ongeveer 0.2 milliseconden per afbeelding, en het testen duurde ongeveer 0.19 milliseconden per afbeelding.



Afbeelding 6

Conclusie en Discussie

Uit het eerste experiment blijkt dat de gemiddelde loss winst tussen neuronen aantallen al snel afdaalt. Ik denk dat dit komt omdat er maar zoveel patronen zijn die een simpel neuraal netwerk kan herkennen in 1 afbeelding. Meer neuronen toevoegen gaat dan na een bepaald punt niet even goed meer helpen.

Ik denk dat de reden dat lagere nudge factor betere resultaten gaven komt, omdat ze voorzichtigere gradient descent stappen maken. Doordat de hogere nudge factor dit niet doen, schommelt de cost functie om het lokale minimum waar deze naartoe gaat. Dit zou een hogere nudge factor in theorie wel beter geschikt maken voor kleinere training data sets. Aangezien een neurale netwerk dan met even veel training afbeeldingen relatief sneller het lokale minimum bereikt.

Tot mijn verbazing bleek uit het derde experiment dat een tweede hidden layer toevoegen het neurale netwerk niet verbeterde. Maar dit kan ook een probleem zijn met het iteratief verbeteren op basis van voorgaande testen; sommige parameters hebben ingewikkelde interacties met elkaar. De interacties tussen neuronen met meerdere hidden layers is het beste voorbeeld hiervan. Daarom is het misschien beter om deze interacties in vervolg apart te meten van de rest. Er is ook nog een kans dat deze geteste neurale netwerk toevallig een wat hogere lokale minimum hadden gevonden in de cost functie. De schaal van een neuraal netwerk zorgt voor dit soort onzekerheden als het niet expliciet onderzocht wordt. Een goed vervolg op deze vindingen dus.

Een extra probleem met dit experiment is dat de neurale netwerken die behandeld werden erg oud zijn. De experimenten in dit verslag zouden waarschijnlijk beter gedaan kunnen worden op moderne neurale netwerk structuren zoals convolutionele neurale netwerken.

Literatuurlijst

Banoula, M. What is Cost Function in Machine Learning. Simplilearn.com. https://www.simplilearn.com/tutorials/machine-learning-tutorial/cost-function-in-machine-learning#:~:text=Gradient%20Descent%20is%20an%20algorithm,reach%20the%20least%20possible%20error.

GLEW: The OpenGL Extension Wrangler Library. https://glew.sourceforge.net/

An OpenGL library. GLFW. https://www.glfw.org/

GitHub - nothings/stb: stb single-file public domain libraries for C/C++. GitHub. https://github.com/nothings/stb

MNIST as .jpg. Kaggle. https://www.kaggle.com/datasets/scolianni/mnistasjpg

Extra:
3Blue1Brown. But what is a neural network? | Chapter 1, Deep learning [Video]. YouTube. https://www.youtube.com/watch?v=aircAruvnKk

Bijlage

1: Zie NeuralNetwork.h, NeuralNetwork.c en GLMath.h in het code mapje onder headers en src.
2: Zie planning in het planning mapje.
3: Zie github (https://github.com/YoeyT/Eindopdracht-vision).
4: Zie StateMachine.c, StateMachine.h, Statictics.h en Statistics.c in de code.
5: dit document.
6: Zie nogmaals StateMachine.c, StateMachine.h, Statictics.h en Statistics.c in de code.