Neurale netwerken

WISB256 – Programmeren in de Wiskunde

7 maart 2017

Samenvatting

Neurale netwerken worden gebruikt om relaties tussen gegeven in en uitvoer te *leren*. In deze opdracht ga je een klasse schrijven om een neuraal netwerk te representeren en methoden implementeren om zo'n netwerk te trainen op gegeven in en uitvoer.

Kernwoorden: neurale netwerken, automatische differentiatie, optimalisatie.

1 Inleiding

Neurale netwerken worden gebruikt om relaties tussen gegeven in en uitvoor te leren om zo de uitkomst voor nog niet eerder geziene input te kunnen *voorspellen*. Bijvoorbeeld, we willen aan de hand van een aantal voorbeelden handgeschreven cijfers leren herkennen. De invoer bestaat in dit geval uit een set van n pixels $x \in \{0,1\}^n$ met waarde 0 of 1 en de uitvoer bestaat uit een cijfer $y \in [0,9]$.

Het eenvoudigste neurale netwerk is de zogenaamde perceptron. Hier is de invoer direct verbonden met de uitvoer via een zogenaamde *activatiefunctie*, die bepaald of de invoer wel of niet wordt doorgegeven. Het idee is dat de invoer slechts word doorgegeven als een bepaalde drempelwaarde wordt overschreden. Een voorbeeld van zo'n netwerk is te zien in figuur 1. De uitvoer is in dit geval gegeven door

$$y = \sigma(\mathbf{w} \cdot \mathbf{x} + b),$$

waar $\sigma(x) = \frac{1}{1+e^{-x}}$ de activatiefunctie is, $\mathbf{w}$ de vector met gewichten, b is de bias en $\mathbf{x}$ de vector met inputs.

Om het neurale netwerk te kunnen gebruiken moeten we het eerst trainen, wat in dit geval betekend dat we de parameters $\mathbf{p} = (\mathbf{w}, b)$ vinden zodat het netwerk in ieder geval een juiste voorspelling geeft op een aantal voorbeelden met de bijbehorende uitvoer. We kunnen dit doen door de parameters te vinden waarvoor de fout zo klein mogelijk is. We meten deze fout met de functie f

$$f(\mathbf{p}) = \sum_{i=1}^{m} (\sigma(\mathbf{w} \cdot \mathbf{x}_i + p) - y_i)^2,$$

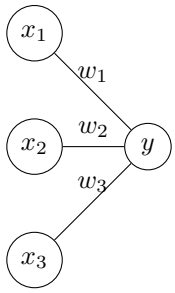
waar x_i de invoer van het i-de voorbeeld is en y_i de bijbehorende uitvoer.

Het vinden van de parameters die deze functie minimaliseren doen we door middel van het *steepest descent* algoritme, wat als volgt werkt. Begin met een schatting van de parameters, bijvoorbeeld $\mathbf{p} = (1,1,1,0)$ en bereken de partiële afgeleiden $\nabla f(\mathbf{p}) = (\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \ldots, \frac{\partial f}{\partial w_k}, \frac{\partial f}{\partial b})$. Van Infi weten we dat deze gradient wijst in de richting waarin f het snelst toeneemt. De richting waarin de functie het snelst afneemt is dus $-\nabla f$. Als we w nu een klein stapje in die richting aanpassen, komen we dus dichter bij het minimum. In de praktijk herhalen we dit een aantal keer:

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} - \alpha \nabla f(\mathbf{p}^{(k)}),$$

waarbij α de stapgrootte is.

Als we eenmaal de parameters hebben gevonden kunnen we ons netwerk gebruiken om de uitvoer te voorspellen voor invoer die we nog niet gezien hebben.



Figuur 1: voorbeeld van een neuraal netwerk met 3 input nodes en een output.

Met een eenvoudige implementatie van een netwerk met 2 invoer en 1 uitvoer kunnen we bijvoorbeeld een logische OF operatie nabootsen. In dit geval hebben we 3 parameters en $m = 4$ voorbeelden $\mathbf{x}_1 = (0, 0)$, $\mathbf{x}_2 = (1, 0)$, $\mathbf{x}_3 = (0, 1)$ en $\mathbf{x}_4 = (1, 1)$ en de bijbehorende uitvoer $y_1 = 0$, $y_2 = 1$, $y_3 = 1$ en $y_4 = 1$. In onze code gaan we gebruik maken van `numpy` om met vectoren te kunnen rekenen.

2 De opdracht

Voor deze opdracht ga je code schrijven om een perceptron te trainen en te gebruiken om voorspellingen te doen.

- Maak een klasse `NeuralNetwork` met methoden als `train` om bijvoorbeeld de parameters te bepalen voor gegeven invoer $\{x_i, y_i\}$ en `predict` om de output y te voorspellen voor een geven input x. De klasse moet de optimale parameters onthouden als het netwerk eenmaal is getraind
- Train een netwerk om bijvoorbeeld een simpele logische operatie te simuleren (OF, EN, ...)
- Test je code op de bijgeleverde Iris dataset
- Breid je code uit om ook netwerken met meerdere lagen te ondersteunen.
- Om de gradienten uit te rekenen kun je ook gebruik van een Python bibliotheek genaamd `autograd`, die je kunt downloaden van het internet: `https://github.com/HIPS/autograd`.
- Test je code op de MNIST dataset: `http://yann.lecun.com/exdb/mnist/`.