Aufgabenblatt 3: Optimierung von Deep Learning Modellen

Aufgabe 1: Grundelemente

Drei Optimierungsebenen: die allgemeine **Struktur eines Deep Learning Netzwerkes**. Z.B. die Anzahl der Nodes, Hidden Layers, Batches und Epochen können optimiert werden. Zudem hängt der Erfolg des Netzes von den gewählten Aktivierungsfunktionen (in den Hidden Layers und der Outputlayer) ab. Im Bereich **der Back Propagation** wird der Optimierer, Batch Normalization (Standardisierung der Inputs in die nachfolgende Layer) und die Lernrate gewählt. Grundsätzlich wird bei der Lernrate klein gestartet, denn wenn die Schritte zu gross sind, kann es sein, dass das Gradientenverfahren nicht greift. Um **Overfitting** zu vermeiden können ganze Knoten gedroped oder höhere Gewichte bestraft werden (Regularization).

Back Propagation: Der Trainingsdatensatz wird nach jedem Durchlauf mit den tatsächlichen Werten verglichen. Basierend auf der Fehlerberechnung wird das Netzwerk in einem Rückwärtsdurchlauf optimiert. Dabei werden die Einflüsse der einzelnen Gewichte berechnet und optimiert. Grundsätzlich wird mittels der Gradientenverfahrens nach dem Minimum der Fehlerfunktion gesucht.

Overfitting: Es ist wichtig, das Netzwerk nicht zu komplex und auf die Trainingsdaten angepasst zu gestalten (ohne bei den Testdaten gut zu sein). Um dies zu vermeiden, können höhere Gewichte bestraft werden, weniger Durchläufe erlaubt werden oder ganze Knoten (Neuronen) während eines Durchlaufs deaktiviert werden (Dropout).

Aufgabe 2: Boston Housing

Im Vergleich zum Antwortblatt 2, habe ich bereits beim Erstellen des Base Model zwei Anpassungen vorgenommen: Die Output Aktivierung habe ich auf «linear» gesetzt und die Metrik auf «mae» damit diese sich von der Loss-Funktion unterscheidet.

Das Ganze Modell habe ich dann nach den Batch Sizes, Epochen, Hidden Layers, Anzahl Knoten etc. optimiert, indem ich getestet habe, welche Grössen das beste Resultat erzeugen. Die habe ich mehrfach durchgespielt, bis ich auf folgende Resultate gekommen bin.

Batch Size: Dies zuerst in einem Range bis zu 128 (in 16er Schritten, da der Computer so aufgebaut ist). Bei einer Batch Size unter 32 ist der MAE von Anfang an relativ tief und springt danach auch nicht mehr. ➔ Batch Size wird auf 16 gesetzt.

Anzahl Epochen: Da bereits fast von der ersten Epoche an, ein gutes Resultat ausmachbar ist, reduziere ich die Anzahl Epochen. ➔ Anzahl Epochen 20.

Anzahl Hidden Layers: es werden bis zu 5 Hidden Layers mit je 12 Knoten hinzugefügt (kleiner als beim Inputlayer). Ab zwei Hidden Layers ist das Resultat bereits gut (MAE tief). ➔ Hidden Layers = 2

Anzahl Knoten: Da im Inputlayer nur 13 Knoten sind, wird getestet wie sich das Modell verhält mit der Anzahl Knoten zwischen 2 und 12 ausgewertet (in Zweierschritten) ➔ Nodes werden auf 6 gesetzt. Da sie bei 8 ziemlich springen und bei 2 und 4 im ersten Moment schlechter werden.

Aktivierungsfunktion der Hidden Layers: Relu, Sigmoid und Tanh wird getestet. ➔ Relu wird beibehalten (von Anfang an konstantes Ergebnis).

Initial Weight: vier Optionen werden getestet (random normal, random uniform, zeros, ones). Wobei die Ones nicht so gut performen wie die restlichen. ➔ Random normal wird belassen.

Optimizer: Die vier mir bekannten Optimizer werden getestet. ➔ Adam performet von Anfang an am besten.

Lernraten zwischen 0.001 und 0.05 werden getestet → Lernrate kann auf 0.01 erhöht werden.

Regularization: es wird getestet, wie sich das Modell verhält bei der l1, l2, l1 und l2, so wie keiner Regularisierung. Alle haben nach 10 Epochen dasselbe Resultat. ➔ keine Regularisierung.

Dropout: Es werden Dropout Raten zwischen 0 und 0.5 getestet. Nach sieben Epochen schwanken alle Modelle im Rahmen von 0.01 MAE. Dropout Rate hat somit keinen Einfluss. ➔ Dropout Rate = 0

Gesamthaft ist das optimierte Modell im Vergleich zum Ursprungsmodell bereits nach wenigen Epochen auf einem tiefen, konstanten MAE. Zudem ist sowohl der Loss-Score, wie auch der «MAE-Score» tiefer im optimierten Modell. Weshalb ich die Optimierung für gelungen halte.

Aufgabe 3: Interpretation Model MNIST

In jedem Layer werden alle Knoten mit allen Knoten der vorhergehenden Layer verbunden und mit einem Bias addiert. Deshalb berechnet sich die Anzahl der zu berechnenden Parameter pro Layer wie folgt: Anzahl Knoten vorhergehender Layer * Anzahl Knoten Layer + Anzahl Bias (gleich wie Anzahl Knoten).

Bei den drei Modellen ist dies wie folgt

Inputknoten sind bei allen 784 Knoten (28*28)

	Erstes Modell	**Zweites Modell**	**Drittes Modell**
Hidden Layer 1	784*512+512 = 401‘920	307‘720	307‘720
Hidden Layer 2	512*8+8 = 4‘104	77‘028	38‘514
Hidden Layer 3		19‘306	990
Hidden Layer 4		4‘752	
Output Layer	8*10+10 = 90	490	
Gesamte Anzahl zu berechnender Parameter	**406‘114**	**409‘296**	**347‘224**

Obwohl im zweiten Modell leicht mehr Parameter berechnet wurden. Wurde doch nur die Hälfte der Zeit pro Epoche benötigt. Das heisst, die Performance ist davon abhängig auf wie viele Layer die Parameter verteilt sind.

Das Modell mit der kleinsten Anzahl zu berechnender Parameter, hat auch die schlechtesten Resultate geliefert. Wobei dies natürlich nicht nur von den Anzahl Layer und Knoten abhängig ist, sondern wie oben beschrieben, von vielen Parametern wie z.B. Batch Size, Regularization oder Anzahl Epochen abhängig sein kann.