Lillis Gehirn – Dokumentation zur Datenverarbeitung von MNIST-Datenbanken

1. Ziel

Ziel war es, ein Netz zur Klassifizierung von MNIST-Datenbanken zu erstellen. In dieser Arbeit wurde sich dabei hauptsächlich darauf konzentriert die Accuracy zu steigern.

2. Code Grundlage

Quelle: How to Develop a CNN for MNIST Handwritten Digit Classification - MachineLearningMastery.com

Als Grundlage des Projektes dient der vom Dozent zur Verfügung gestellte Code des Jupyter Notebooks. In diese Grundlage wurde das Modell aus der genannten Quelle eingearbeitet. Dies stellt die Code-Grundlage für die Arbeit auf die im Folgenden aufgebaut wird.

Total params: 216854

Best val_accuracy: 0.9938

3. Accuracy steigern

3.1. Dropout

Quelle: srivastava14a.pdf (jmlr.org)

Dropout deaktiviert während des Trainings einzelne Einheiten des Netzwerks. Dadurch müssen die verbleibenden Einheiten mit verschiedenen anderen Elementen zusammenarbeiten. Dadurch kann Overfitting vermieden werden.

Laut Quelle ist eine Dropoutrate von 20% bei den Eingabe-Layer und 50% bei den Hidden-Layer optimal.

Anwendung der Droput-Metode auf den MNIST-Datensatz:

Fehlerquote wird bei Netzen mit Dropout von 1,6% auf 1,35% im Vergleich zu Netzen ohne Dropout abgesenkt. Durch die RELU Funktion kann auf 1,25% Fehlerquote minimiert werden. Mit May-Norm-Regularisierung auf 1,06%.

Ergebnisse:

Durch Einbau von drei Dropout-Layern mit Dropoutrate 0.25 bei der Eingabe-Layer und 0.5 bei Hidden-Layer und Max-Norm-Regularisierung:

Total Params: 216854

Val_accuracy: 0.9955

Einbau von Dopout RBM: `DropoutRBM(num_hidden=100, dropout_rate=0.25),`

Total Params: 1020922

Val_accuracy: 0.1135

➔ Ansatz wurde sofort!!! Wieder entfernt

3.2. Learning Rate anpassen

Quelle: Going beyond 99% — MNIST Handwritten Digits Recognition | by Jay Gupta | Towards Data Science

Learning rate: Wenn erkannt wird, dass der Lernfortschritt des Modells stagniert, wird die learning rate dynamisch verändert um bessere Ergebnisse zu erreichen. Es wurden verschiedene Werte für factor und patience ausprobiert, das beste Ergebnis gab es bei einem Faktor von 0,2, einer patience von 1,5 und einem minimalen Delta von 1E^-7.

Total Params: 216854

Val_accuracy: 0.9959

3.3. Batch Normalization

Quelle: Going beyond 99% — MNIST Handwritten Digits Recognition | by Jay Gupta | Towards Data Science

Die Batch-Normalization Layer wird nach jedem Conv-Layer und vor jeder Activation Layer eingefügt. Dadurch werde die Aktvierungen normalisiert, das Training beschleunigt und die Lernraten erhöht.

Total Params: 217638

Val_accuracy: 0.9964

3.4. Regularization

Quelle: Regularization Techniques And Their Implementation In TensorFlow(Keras) | by Richmond Alake | Towards Data Science

L1 Regularisierung:

Gewichtswerte, die nahe an null oder negativ sind haben kaum Auswirkung auf die Gesamtleistung des Modells und werden daher auf null gesetzt. Dadurch kann Speicherplatz gespart werden.

➔ Die L1 Regularisierung hat in meinem Modell negative Auswirkungen auf die Accuracy gezeigt, deshalb wurde sie entfernt

L2 Regularisierung:

Transformiert Gewichtswerte zu einem Wert näher zu 0, sodass eine einheitlichere Gewichtsverteilung vorliegt. Dadurch werden einzelne, sehr große Gewichtswerte verhindert, was wiederrum Overfitting verhindert.

➔ Die L2 Regularisierung wurde mit einer Rate von 0.0005 in zwei Convolutional Layer eingebaut
➔ In diesem Schritt wurde außerdem eine weitere Convolutional Layer mit 32 Filtern eingebaut, was die Parameterzahl verringert hat.
➔ **Total Params: 169286**
Val_accuracy: 0.9969