

DeepLearning

Tim Lucas Halt

8. Dezember 2023

Sehr kurz

1 Methodik

Mein Ziel war in einen der drei vorlesungsbegleitende Benchmarks zu führen und die Anstrengungen vorerst auf dieses Ziel zu legen.

Dafür habe ich mich in der ersten Woche in alle drei Richtungen, maximale Genauigkeit, minimale Labels und minimale Parameter probiert.

Zielerreicht wurde mit dem Netzt welches in Abschnitt 2 vorgestellt wurde. Eine kombination von verschiedensten quellen zusammengedrafen und mit etwas glück auf 99,47 % gebracht

Die weitere Verbesserung wollte ich vor allem automatisieren.

Dafür habe ich Kostenlose Grafikkarten von Kaggle (30 Stunden die Woche) und Colab (12 Stunden am Tag) genutzt.

Im ersten Schritt wollte ich die Wahl der Hyperparemter meines optimieren. dafür habe ich verschiedene Kombinationen mit weights & biases sweepen und tracken lassen

2 Basis -

Das Ausgangsnetz von dem die Untersuchungen ausgehen ist in Listing 1 dargestellt. Es hat etwa 62-tausend Parameter und wurde mit dem vollständigen Trainingsdatensatz (60000 Label) trainiert.

Kompiliert mit dem Adam Optimizer mit einer Learning rate von 0.003 Mit einer Batch-Size von 512 erreichte das Modell nach 100 Epochen 99,47 % val_accuracy

Listing 1: Python-Code

```
1 InputLayer(input_shape=(28,28,1)),
2 Conv2D(filter=28, kernel_size=5, padding='same', activation='relu'),
3 MaxPooling2D(2,2),
4 Conv2D(filter=16, kernel_size=5, padding='same', activation='relu'),
5 MaxPooling2D(2,2),
6 layers.Dropout(0.2),
7 layers.Flatten(),
8 layers.Dropout(0.2),
```

```

9 layers.Dense(64, kernel_regularizer = tf.keras.regularizers.l2(0.07),
  activation = 'relu'),
10 GaussianNoise(0.1),
11 Dense(10, activation='softmax')
```

Erkenntnis learning rate und batchsize haben einfluss, aber wie?

Sweep, um filteranzahl in conv2d layer zu testen, sowie die des zweitletzten dense layers zu verbessern und batch size zu optimieren

3 N

V02 – Anpassung und Erweiterung Anpassung der Filter- und Neuronen-Anzahl Ergänzung um BatchNormalization nach 4)

Reihenfolge Batch, MaxPool, Dropout

Activation Test

4 wandb

Zunächst erfolgte die Untersuchung der Filter-Anzahlen in den beiden Convolution-Layern und der Neuronen-Anzahl des ersten Dense-Layer. Dafür wurde die Sweep-Funktion von Weights & Biases verwendet (<https://docs.wandb.ai/guides/sweeps>). Das Output-Layer beinhaltet in allen folgenden Netzen 10 Neuronen und die *softmax*-Aktivierungsfunktion, damit jeder Klasse (bzw. Zahl) eine Wahrscheinlichkeit zugeordnet wird.

Die Wahl der Kombinationen erfolgt mit der Methode *bayes* zur Maximierung der *val_acc*, die aus einem vorgegebenen Raster an Werten versucht eine Verbesserung zu erreichen. Das Raster wurde nach drei Kriterien. Erstens sollte eine Abstufung im Convolution-Teil erfolgen (dieses vorgehen erwies sich mit der Zeit als kontraproduktiv und wurde verworfen). Zweitens sollten nicht nur 2er-Potenzen zur Verfügung stehen. Und letztens sollte die Neuronen-Anzahl im Dense-Layer eine größere Auswahl haben, weil die Meinungen in Foren diesbezüglich streut.

Die Ergebnisse sind in Abbildung 1 dargestellt. Hervorgehoben sind die besten zehn Kombinationen. Die *val_acc* war hoch, für eine Filteranzahl der Convolution-Layers

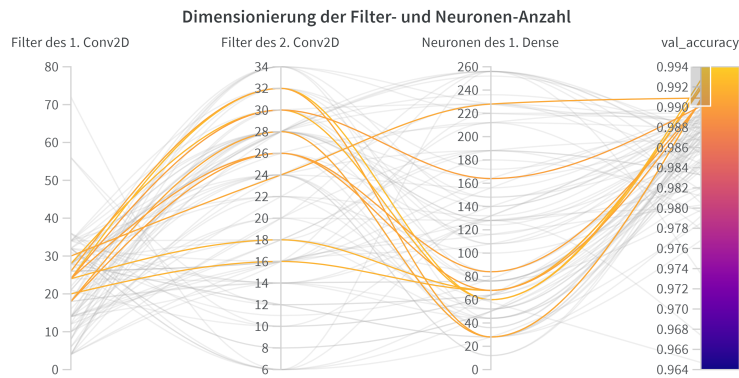


Abbildung 1: 2

um den Wert 28. Für die Neuronen des ersten Dense-Layers konzentrieren sich der erfolgreichen Kombinationen bei den kleineren verfügbaren Werten.

Für die Wahl wurden noch weitere Kombinationen in den eingegrenzten Bereich getestet. Schlussendlich erwiesen sich 28 Filter in beiden Convolution-Layern als erfolgreich, wodurch das Symmetrie-Bedürfnis wegen der Ähnlichkeit zur Bildgröße von 28x28 Pixeln kurzzeitig befriedigt wurde. Die Befriedigung hielt nur kurz an, da sich für die Neuronen-Anzahl 54 etablierte, was weder eine 2er-Potenz noch ein Vielfaches von 28 ist.

bei unterschiedlichen batchsize und learning rate bayes max val:acc

anders als bei den aktivierungsfunktionen. Hier zeigte sich eine gute Genauigkeit bei der Sigmoid-funktion im

Gandalf in Moria: „Im Zweifelsfall sollte man immer seiner Nase folgen ...“

Aus insgesamt 250 Kombinationen stellte

Gandalf in Moria: „Im Zweifelsfall sollte man immer seiner Nase folgen ...“

umso enger die Auswahl an Hyperparametern wurde, umso mehr Epochen bis zum Vergleich

Untersuchungen der Dropoutwerte zeigten Erfolg 0,4 und GaussianNoise 0.75. Hinsichtlich des Papers (Dropout gegen Overfitting)

Sonstige Versuche: Adam mit SGD ersetzen erfolglos

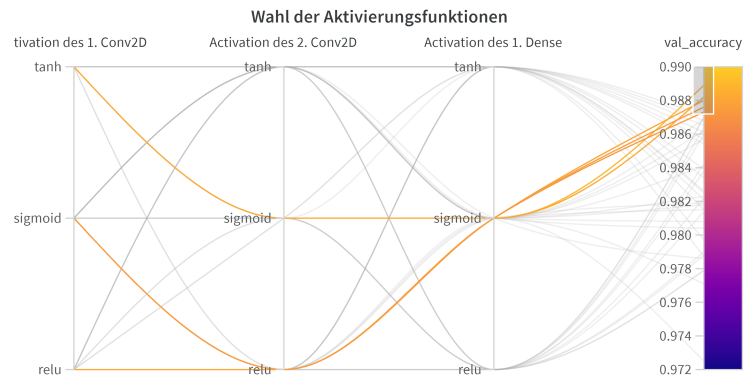


Abbildung 2: 1

dropout:

```

1 tf.keras.layers.InputLayer(input_shape=(28,28,1)),
2 tf.keras.layers.Conv2D(filters=neurons1, kernel_size=5, padding='same',
   activation=activation1),
3 tf.keras.layers.BatchNormalization(),
4 tf.keras.layers.MaxPooling2D(2,2),
5 tf.keras.layers.Dropout(do1),
6 tf.keras.layers.Conv2D(filters=neurons2, kernel_size=5, padding='same',
   activation=activation2, kernel_initializer=tf.keras.initializers.
   HeNormal()), #new
7 tf.keras.layers.MaxPooling2D(2,2),
8 tf.keras.layers.Dropout(do2),
9 tf.keras.layers.Flatten(),
10 tf.keras.layers.Dropout(do3),
11 tf.keras.layers.Dense(neurons3, kernel_regularizer = tf.keras.
   regularizers.l2(0.07), activation = activation3),
12 tf.keras.layers.BatchNormalization(),
13 tf.keras.layers.Dropout(do4),
14 tf.keras.layers.GaussianNoise(gn1),
15 tf.keras.layers.Dense(10, activation='softmax')
```

5 Finale - Mit Sinn und Verständnis

6 Optimierung -

Mal schauen was so kommt

Listing 2: Python-Code

```
1  from scipy.spatial import distance
2
3  x_train_best = [] # Array für die ähnlichsten Bilder
4  y_train_best = [] # Array für die zugehörigen Labels
5  similar_indices = [47647] # Array für die Indizes der ähnlichsten
    Bilder
6
7  # Berechnung der ähnlichsten Ziffern für jede Klasse von 0 bis 9
8  for digit in range(10):
9
10     print('Digit:', digit)
11     # Filtern der Ziffern nach ihrer Klasse
12     class_images = x_train[y_train == digit]
13
14     # Berechnung der durchschnittlichen Cosinus-Ähnlichkeit für jede
        Ziffer zu anderen Ziffern derselben Klasse
15     similarities = []
16     for i, image in enumerate(class_images):
17         avg_similarity = 0
18         for other_image in class_images:
19             if not np.array_equal(image, other_image):
20                 # Umwandlung von 28x28 Bildern in Vektoren für Cosinus-Ähnlichkeit
21                 image_vector = image.flatten()
22                 other_image_vector = other_image.flatten()
23                 # Berechnung der Cosinus-Ähnlichkeit
24                 cosine_similarity = 1 - distance.cosine(image_vector,
                    other_image_vector)
25                 avg_similarity += cosine_similarity
26                 avg_similarity /= len(class_images) - 1 # Durchschnittliche Ä
                    hnlichkeit zu allen anderen Ziffern der Klasse außer sich selbst
27                 similarities.append((i, avg_similarity))
28
29     # Sortieren nach der durchschnittlichen Ähnlichkeit und Auswahl der ä
        hnlichsten Ziffer
30     similarities.sort(key=lambda x: x[1], reverse=True)
31     most_similar_index = similarities[0][0]
32
33     most_similar_index_train_images = np.where((y_train == digit))[0][
        most_similar_index]
```

```

34 most_similar_digit = x_train[most_similar_index_train_images]
35
36 print('Index:', most_similar_index_train_images)
37
38 # Hinzufügen des ähnlichsten Bildes, seines Labels und seines Index im
   train_images Array in den Arrays
39 x_train_best.append(most_similar_digit)
40 y_train_best.append(digit)
41 similar_indices.append(most_similar_index_train_images)
42
43 # Umwandeln der Listen in numpy arrays
44 x_train_best = np.array(x_train_best)
45 y_train_best = np.array(y_train_best)
46 similar_indices = np.array(similar_indices)

```

24C5 means a convolution layer with 24 feature maps using a 5x5 filter and stride 1
 24C5S2 means a convolution layer with 24 feature maps using a 5x5 filter and stride 2
 P2 means max pooling using 2x2 filter and stride 2 256 means fully connected dense
 layer with 256 units

784 - [32C3-32C3-32C5S2] - [64C3-64C3-64C5S2] - 128 - 10

7 batchsize

während des testens 256 sinnvoll gleicher erfolg auch mit 32 aber mehr streuung und
 hinsichtlich banchmark peaks gewünscht

8 callbacks

9 Dropout

Paper: Dropout: A Simple Way to Prevent Neural Networks from Overfitting Nitish
 Srivastava nitish@cs.toronto.edu Geoffrey Hinton hinton@cs.toronto.edu Alex Kriz-
 hevsky kriz@cs.toronto.edu Ilya Sutskever ilya@cs.toronto.edu Ruslan Salakhutdi-
 nov

10 Variable Learning Rate

Listing 3: Variable Learning-Rate

```
1     vlr = tf.keras.callbacks.ReduceLROnPlateau(monitor='loss', factor =  
      0.95, patience=10, min_lr=0.0001)  
2     vlr2 = tf.keras.callbacks.LearningRateScheduler(lambda x: 1e-3 *  
      0.995 ** x)
```

11 Optimierung

Chat GPT: Wenn du zwei aufeinanderfolgende Convolutional Layers mit einer Filtergröße von 3x3 verwendest, wird das Netzwerk in der Lage sein, komplexere und nichtlineare Muster zu erlernen. Hier sind ein paar Gründe, warum das der Fall ist:

Receptive-Field-Erweiterung: Die Verwendung von zwei Convolutional Layers mit einer Filtergröße von 3x3 führt zu einem größeren Receptive Field im Vergleich zu einer einzelnen Schicht mit 5x5 Filtergröße. Dadurch kann das Netzwerk über einen größeren Bereich des Eingabebildes Informationen erfassen.

Mehrere Nichtlinearitäten: Jede Convolutional Layer wird durch eine nichtlineare Aktivierungsfunktion wie ReLU aktiviert. Wenn zwei 3x3 Convolutional Layers hintereinander geschaltet werden, werden zwischen den Schichten zwei Aktivierungsfunktionen angewendet, was zu einer stärkeren nichtlinearen Transformation der Eingabedaten führt.

Parameterreduktion: Zwei 3x3 Convolutional Layers haben insgesamt weniger Parameter als eine 5x5 Convolutional Layer. Weniger Parameter bedeuten weniger Berechnungen und weniger Anfälligkeit für Overfitting, da weniger spezifische Features gelernt werden müssen.

Lernen von tieferen Merkmalen: Das Stapeln von mehreren Convolutional Layers hintereinander ermöglicht es dem Netzwerk, hierarchische und abstrakte Merkmale zu lernen. Durch die schrittweise Verfeinerung der Merkmale von einer Schicht zur nächsten können komplexere Repräsentationen des Eingaberaums erzeugt werden.

Dieses Konzept der Verwendung von kleineren Filtern nacheinander anstelle eines einzelnen großen Filters wird oft in modernen Convolutional Neural Networks verwendet, um die Lernfähigkeit und Effizienz zu verbessern. Es trägt dazu bei, dass das Netzwerk tiefergehende und komplexere Muster im Eingaberaum erfassen kann.

<https://doi.org/10.48550/arXiv.2202.01653>

Chat GPT: Strided Convolution für Downsampling: Bei Verwendung einer Convolutional Layer mit Strides von 2 werden die Ausgabe-Feature-Maps aufgrund des größeren Schrittweitenwerts um den Faktor 2 in jeder Dimension reduziert. Das führt zu einer räumlichen Reduzierung der Feature-Maps und erzeugt somit einen Downsampling-Effekt ähnlich dem einer Max-Pooling-Schicht.

Trainierbare Downsampling-Operation: Im Gegensatz zur Max-Pooling-Schicht, die eine festgelegte Operation (Maximum aus einem Fenster) ohne trainierbare Parameter ist, ermöglicht die Verwendung einer Convolutional Layer mit Strides von 2 das Lernen von Downsampling-Operationen. Die Gewichte in der Convolutional Layer werden während des Trainings angepasst, um die beste Darstellung der Daten zu finden, während gleichzeitig eine Reduzierung der räumlichen Dimensionen erfolgt.

End-to-End-Lernen: Durch die Verwendung einer Convolutional Layer für Downsampling wird das Lernen von Merkmalen und Downsampling in einem End-to-End-Modell ermöglicht. Das Netzwerk lernt die geeigneten Filter, um Merkmale zu extrahieren, während es gleichzeitig die räumliche Dimension reduziert, ohne auf separate Schichten wie Max-Pooling zurückgreifen zu müssen.

Flexibilität und Anpassungsfähigkeit: Die Convolutional Layer mit Strides von 2 bietet Flexibilität in Bezug auf die Art des gelernten Downsampling. Das Netzwerk kann verschiedene Arten von Downsampling lernen, die möglicherweise besser zur spezifischen Datenverarbeitung passen als die festgelegte Max-Pooling-Operation.

Die Verwendung einer Strided Convolution für Downsampling bietet somit die Möglichkeit, eine lernbare Downsampling-Operation zu integrieren, die besser in das Gesamtkonzept des neuronalen Netzes eingebunden werden kann und es dem Netzwerk ermöglicht, sowohl Merkmale zu lernen als auch die Dimensionen der Feature-Maps zu reduzieren.

12 Sonst noch

cosinus nächste

Autoencoder