TensorFlow Merkblatt-1/2

www.datamics.com

Grundlagen

TensorFlow ist ein frei verfügbares Framework für dataflow programming und wird insbesondere im machine learning eingesetzt, beispielsweise zur Spracherkennung.

Installation

```
pip3 install –upgrade pip
pip3 install numpy scipy joblib keras \
  tensorflow
pip3 install -U scikit-learn
```

Imports

```
import numpy as np
import nputils
import tensorflow as tf
from sklearn.datasets import load_iris
from keras.utils import to_categorical
from sklearn.model_selection import \
  train_test_split
from sklearn.preprocessing import \
  MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, DropOut
from sklearn.metrics import \
  confusion_matrix, classification_report
from keras.models import load_model
from keras.datasets import mnist
```

Keras Iriskategorie I

Keras ist eine frei verfügbare Bibliothek mit Komponenten für den Entwurf komplexer NNs und wird häufig mit TensorFlow eingesetzt.

Datensatz laden

```
iris = load_iris()
print(iris.DESCR)
X = iris.data
y = iris.target
y = to_categorical(y)
```

Trainingsdaten aufteilen

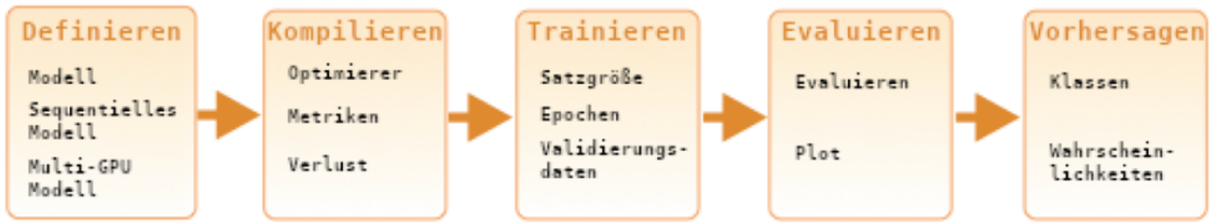
```
X_train, X_test, y_train, y_test = \
  train_test_split(X, y, test_size=0.33,
    random_state=42)
```

test_size ist der Anteil der exklusiv zur Evaluierung genutzten Daten und **random_state** der Startwert des Zufallsgenerators.

Daten standardisieren

```
scaler_object = MinMaxScaler()
scaler_object.fit(X_train)
scaled_X_train = \
  scaler_object.transform(X_train)
scaled_X_test = \
  scaler_object.transform(X_test)
```

Optimierung mit NNs



Keras Iriskategorie II

NN mit Keras einrichten

```
model = Sequential()
model.add(Dense(8, input_dim=4,
  activation='relu'))
model.add(Dense(8, input_dim=4,
  activation='relu'))
model.add(Dense(3,
  activation='softmax'))
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy'])
```

Modell trainieren

```
model.fit(scaled_X_train, y_train,
  epochs=150, verbose=2)
```

epochs bestimmt die Anzahl der Durchläufe des Trainings. Werden bei einem Training zu schlechte Resultate erzeugt, kann es sich lohnen, **epochs** hochzusetzen.

Modell speichern und laden

```
model.save('meinmodel.h5')
newmodel = load_model('meinmodel.h5')
newmodel.predict_classes(X_test)
```

Neue Daten klassifizieren

```
model.predict_classes(scaled_X_test)
```

Leistungsevaluation

```
model.evaluate(
  x=scaled_X_test, y=y_test)
predictions = \
  model.predict_classes(scaled_X_test)
y_test.argmax(axis=1)
confusion_matrix(
  y_test.argmax(axis=1), predictions)
print(classification_report(
  y_test.argmax(axis=1),predictions))
```

Layertypen

layer_input()
Eingabelayer

layer_dense()
Hängt ein dichtverknüpftes Layer an die Ausgabe

layer_activation()
Wendet eine Aktivierungs-funktion auf die Ausgabe an

layer_dropout()
Wendet Dropout auf die Eingabe an

layer_reshape()
Formt eine Ausgabe um

layer_permute()
Permutiert die Dimensionen einer Eingabe

layer_repeat()
Wiederholt die Eingabe n mal

layer_lambda(object, f)
Stellt einen beliebigen Ausdruck als Layer dar

layer_activity()
Aktualisiert die auf der Kostenfunktion basierte Eingabeaktivität

layer_masking()
Maskiert eine Sequenz, z.B. nur bestimmte Zeitpunkte

layer_flatten()
Verflacht eine Eingabe

Keras Bilderkennung I

Eingabelayer

Training mit Bildern des MNIST-Datensatzes:

```
(X_train, y_train), (X_test, y_test) = \
  mnist.load_data()
```

Verformen und skalieren

```
X_train = X_train.reshape(
  X_train.shape[0], 1, 28, 28)
X_test = X_test.reshape(
  X_test.shape[0], 1, 28, 28)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
Y_train = np_utils.to_categorical(
  y_train, 10)
Y_test = np_utils.to_categorical(
  y_test, 10)
```

Modell und Layer definieren

```
model = Sequential()
model.add(Convolution2D(
  32, 3, 3, activation='relu',
  input_shape=(1,28,28)))
model.add(Convolution2D(
  32, 3, 3, activation='relu'))
model.add(
  MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10,
  activation='softmax'))
```

Kompilieren

```
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy'])
```

Trainieren

```
model.fit(X_train, Y_train, batch_size=32,
  nb_epoch=10, verbose=1)
```

Evaluieren und Klassifizieren

```
model.evaluate(x_test, y_test))
model.predict_classes(x_test))
```

DATAMICS
machine intelligence consulting services

TensorFlow Merkblatt-2/2

Glossar

Neuronales Netzwerk (NN)

Ein Neuronales Netzwerk besteht aus einer beliebigen Anzahl untereinander verbundener, künstlicher Neuronen, die in Layern organisiert sind. Neuronen stellen dabei die Kanten und deren Verbindungen die Linien im Graph des Netzwerks dar. Dabei ist jede Verbindung mit einer Gewichtung versehen, welches erlernt wird und das Verhalten des Netzwerks bedingt. Dabei existieren ein- oder mehrschichtige NNs, die Informationen nur nach oben weiterleiten (**feedforward**) sowie rekurrente NNs, die Informationen auch nach unten weiterleiten (**feedback**, Rückkopplung).
Einrichtung eines NNs:

Layerstruktur bestimmen
Train (Trainieren)
Evaluate (Evaluieren)
Predict (Vorhersagen)

Layer (Ebene)

Layer werden die Ebenen eines NNs genannt. Die Anzahl der Layers hat einen große Einfluss auf die Komplexität, die ein NN abbilden kann. Beispielsweise ist ein einschichtiges NN nicht dazu in der Lage, das XOR-Problem zu lösen.

Deep Learning

Deep Learning bezeichnet auf NN basierende Optimierungsverfahren mit einer Vielzahl von Layers und Neuronen. Infolge des komplexen erlernten Verhaltens entwickelt das NN eine komplexe Innenstruktur.

Rückverfolgung (Backpropagation)

Bei der Backpropagation wird der errechnete Fehler der Vorwärtspropagation einer bekannten Eingabe durch ein NN anteilig an die beteiligten Neuronen zurückgereicht, die ihre Gewichte entsprechend anpassen.

Train (trainieren)

Der Trainingsdatensatz, der bei Supervised Learning zur Verfügung steht, wird üblicherweise in (mindestens) zwei gleich große Teile aufgeteilt. Der eine Teil wird für das eigentliche Training genutzt, der andere zur Evaluierung der Ergebnisse des Trainings. Beim Training wird die Gewichtung der Verbindungen der Neuronen, z.B. über Backpropagation, auf die Trainingsdaten optimiert.

Recurrent Neural Network (RNN)

Ein RNN nutzt die Ausgabe von Neuronen als Eingaben, um Rückkopplungsprozesse zu schaffen.

Feedback-Typen

direct (direkt)
Ein Neuron nutzt die eigene Ausgabe als Eingabe
indirect (indirekt)
Die Ausgabe eines Neurons führt zu einem Neuron eines vorherigen Layers
lateral
Die Ausgabe eines Neurons führt zu einem Neuron in derselben Schicht
general (allgemein)
Die Ausgabe eines Neurons führt zu allen anderen Neuronen

Kanäle

Weiterhin werden RNNs anhand der Anzahl der Eingabe- und Ausgabekanäle unterschieden:

one-to-one - Bildklassifikation
Eingabebild erzeugt Ausgabeklassifikation
one-to-many - Bildcharakteristiken
Eingabebild erzeugt mehrere Ausgabe-Charakteristiken
many-to-one - Gefühlsanalyse
Serie von Eingabedaten (Text) erzeugt Gefühlsklassifikation
many-to-many - Maschinenübersetzung
Serie von Eingabedaten wird zu einer Serie von Ausgabedaten

Convolutional Neural Network (CNN

Ein CNN ist aus folgenden Layertypen aufgebaut:

Convolutional Layer

Das Convolutional Layer wird auf die Eingabe angewendet und versucht, Features zu finden. Das Ergebniss ist eine Feature Map.

Filtergröße: Größe des Suchfensters
Stride: Größe der Suchfensterverschiebung

Pooling Layer

Vereinfacht die Ausgabedaten der Convolutional Layer, so dass nur wichtige Features verfolgt und gelernt werden.
Pooling-Dimensionen:

1D: Vektoren, z.B. Zeitreihen
2D: Matritzen, z.B. 2-D-Bild Farbraum
3D: Zeit-räumliche Daten, z.B. MRT

Fully connected Layer

Als letzte Ebene folgt das fully connected Layer, welches Eingaben von allen Neuronen des Netzwerks empfängt. Das Fully connected Layer ermittelt beispielsweise den Score einer Bildklassifizierung.

Datentypen

Tensor

Ein Tensor ist der grundlegende Datentyp von TensorFlow und Instanz eines hochdimensionalen Arrays. Tensoren können mit Platzhaltern zwischen Sessions persistent gehalten werden.

Graph

Ein Graph ist eine Liste zum Speichern von Tensoren, Variablen und Operationen. Dabei ist immer nur ein Graph aktiv. Wird ein Optimierer erzeugt und eine Minimierung vorgenommen wird, so wird deren Ergebnis ebenfalls im Graph gespeichert.

Sessions (Sitzungen)

Zwar speichern Graphen Operationen, ausführen können sie diese jedoch nicht. Stattdessen muss eine Session erzeugt und ihre run-Methode mit einem Tensor oder einer Operation aufgerufen werden.

Optimizers (Optimierer)

Optimierung im machine learning bezeichnet die schrittweise mathematische Modellierung eines realen Sachverhaltes mit dem Ziel der genauestmöglichen Näherung. Tensorflow unterstützt vielfältige Optimizer in Form von Klassen, die über Methoden zur Minimierung verfügen.

Variables (Variablen)

Eine Instanz der Klasse Variable dient der Aufbewahrung von Daten während des Optimierungsprozesses, beispielsweise die Form einer Trennlinie in einem Clusterverfahren. Variablen müssen vor ihrer Verwendung initialisiert werden.

Estimators (Schätzer)

Anstatt der Verwendung niedrigleveliger Datenstrukturen wie Session und Graph kann machine learning mit der Estimator-API von Tensorflow durchgeführt werden, die viele verschiedene Estimators bereitstellt. So kann viel Konfigurationsaufwand erspart bleiben. Bei einem Estimator werden immer diese drei Schritte durchgeführt: **train**, **evaluate** und **predict**.

Glossar

Evaluate (evaluieren)

Bei der Evaluierung des Trainings wird die Qualität des Trainings anhand eines nicht für das Training genutzten Teil der Trainingsdaten gemessen. Zur Fehlermessung kann unter anderem die Abweichung zum Mittelwert aller Neuronen, der R2-Score oder bei Clusterverfahren beispielsweise die Homogenität dienen.

Predict (vorhersagen)

Nachdem das NN erfolgreich trainiert und evaluiert wurde, ist es in der Lage, Vorhersagen zu machen. Neue, nicht im Trainingsdatensatz enthaltene Daten werden in das Netz eingespeist, welches eine Klassifikation vornimmt.

Supervised Learning (überwachte Lernverfahren)

Bei Supervised Learning liegen Daten zum Trainieren des NNs bereit. Z.B. kann mit einem großen Datensatz von Bildern ein NN effektiv auf das Erkennen von Hunden in Fotos trainiert werden, da der Erfolg eines Trainingsschrittes klar gemessen werden kann.

Unsupervised Learning (unüberwachte Lernverfahren)

Unüberwachte Lernverfahren haben keine Daten zur Bewertung des Trainingserfolges und dienen der Suche nach Mustern in nichtklassifizierten Daten, beispielsweise bei Clusterverfahren.

Optimierungsverfahren

Softmax

Softmax-Regression (auch multinomiale logistische Regression) ist ein häufig eingesetztes Optimierungsverfahren. Bei Softmax wird der Score der Zugehörigkeit zu einer Kategorie in eine Wahrscheinlichkeit umgewandelt und die Kategorie mit der höchsten Wahrscheinlichkeit als Zuordnung ausgegeben. Damit kann der Fehler eines NNs gemessen werden, woraufhin die Gewichte angepasst werden können.

Adam

Adam (adaptive moment estimation) ist eine für NNs hervorragend geeignete Optimierungsfunktion. Die Lernrate von Adam ist adaptiv und wird an das erste und zweite Momentum des Graphen für jeden Parameter angepasst.