TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Alexander Jenke, Theodor Straube

Übung10: Machine Learning (mit PyTorch)

Programmierkurs Python // Mittwoch, 15. Januar 2020

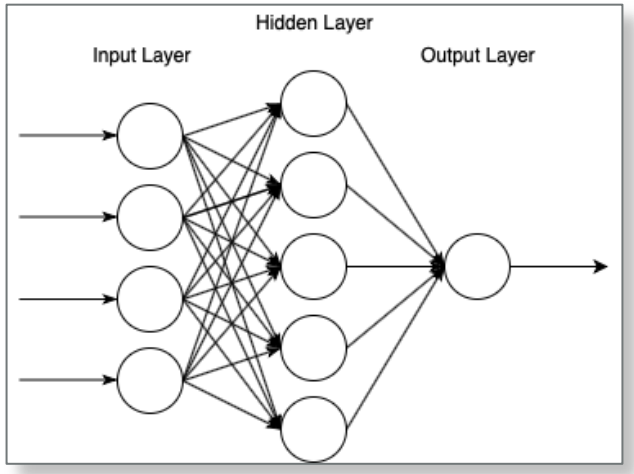
ToC

- Basics
  - Neuronale Netze
  - Frameworks
- PyTorch
  - Tensoren
  - Datasets
  - Training & Evaluation

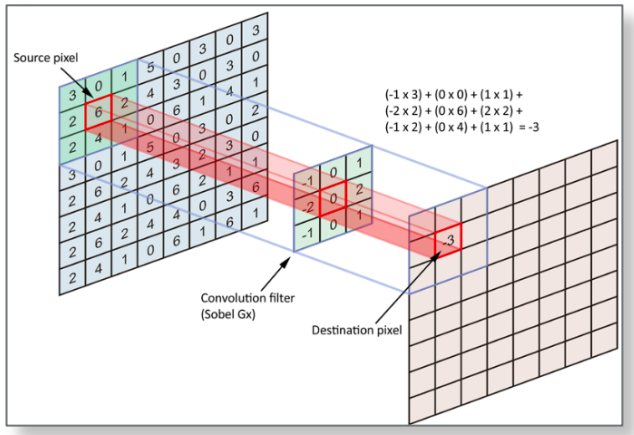
[1] https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

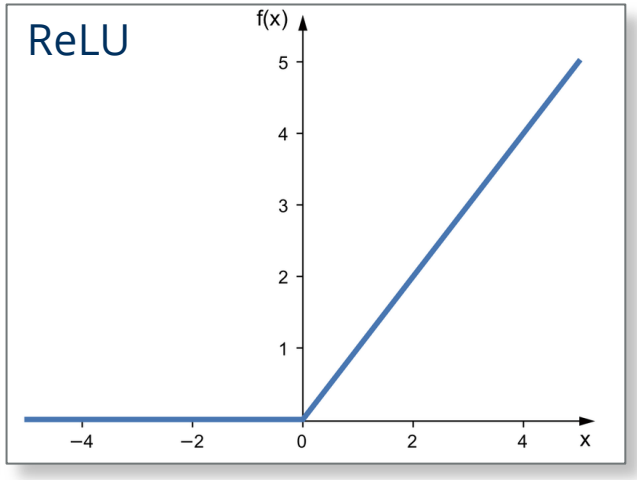
Basics – Neuronale Netze

- Architektur ist angelehnt an das menschliche Gehirn
- Einzelne Nodes summieren eingehende Werte gewichtet auf und geben einen Wert entsprechend ihrer Aktivierungsfunktion weiter
- Im Input Layer werden die zu verarbeitenden Daten eingegeben
- Im Output Layer steht das Ergebnis des Netzwerkes
- Das Hidden Layer lernt die Funktion: Input → Output



- Convolutional Layer filtern 2D Daten mittels eines Kernels
  - ➢ Erkennen von Zusammenhängen in Bildern





Basics – Frameworks

- Tensorflow
- CAFFE
- Amazon Machine Learning (AML)
- Apache Spark
- Scikit-learn
- PyTorch

- Tensorflow eignet sich gut für marktreife Produkte
- Pytorch eignet sich gut für die Entwicklung & Forschung

PyTorch

- Get Started: https://pytorch.org/get-started/locally/
- Doku: https://pytorch.org/docs/stable/index.html
- Installation mit pip: `pip3 install torch torchvision`

- PyTorch ermöglicht:
  - Das bauen von Netzen
  - Das laden von Daten
  - Das weiterreichern von Daten durch das Netz
  - Das berechnen von Losswerten
  - Das zurück propagieren von gradienten
  - Das anpassen des Netzes (das eigentliche Lernen)

PyTorch – Tensor

- Alle Wert in PyTorch werden in sog. Tensoren eingepackt
- Ein Tensor enthält einen oder eine Menge (Listen/Arrays) von Werten
- Der Tensor kümmert sich um das Speichern von Metainfos und Gradienten
- Tensoren unterstützen eine Vielzahl von mathematischen Operationen (+, -, pow, usw.)

- Erstellen: `torch.tensor([1,2,3,4])`
- Den einzelnen Wert bekommen: `Tensor.item()`
- Das Array bekommen: `Tensor.numpy()`

PyTorch – Dataset

- Um eigene Daten in ein Netzwerk zu laden muss eine eigene Dataset-Klasse geschrieben werden
- Die Klasse erbt von torch.utils.data.Dataset
- Die Klasse enthält folgende Funktionen:
  - __init__ : initailisiert die Klasse und bereitet die Daten vor
  - __len__ : gibt einen Integer zurück welcher angibt wie viele Samples existieren
  - __getitem__ : nimmt eine ID & gibt das Sample der übergebenen ID zurück

- Bilder mit PIL & Numpy laden:

```
from PIL import Image
import numpy as np

img = Image.open(fd)
img_np = np.asarray(img.getdata(), dtype=np.float32).reshape((1, 28, 28))
```

PyTorch - Netz

- Netze bestehen aus einzelnen Schichten durch welche die Daten durchgereicht werden
- forward Funktion:
  Kümmert sich um das weiterreichen der Daten durch die einzelnen Schichten

- F.max_pool2d: Halbiert die Bildgröße und nimmt den maximalen der 4 Ausgangswerte
- nn.Conv2d: 2D Convolutional Layer
- nn.Linear: Fully Connected Layer
- nn.Dropout2D:
  setzt zufällig Werte auf 0
  - ➢ Sorgt für ein gleichmäßigeres Training

```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.dropout = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(self.dropout(F.max_pool2d(self.conv2(x), 2)))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

PyTorch – Training & Evaluation

Loss

- Der Loss ist ein Wert wie Falsch das vorhergesagte Ergebnis des Netzes ist
- Der Loss wird durch einen Vergleich der Vorhersage mit dem tatsächlichen Label bestimmt
- Aus dem Loss kann durch zurückpropagieren ein Gradient für jeden Parameter bestimmt werden

- PyTorch stellt gängige Loss-Funktionen zur Verfügung, z.B.: NLLLoss, MSELoss, BCELoss

- Für die MNIST Klassifikation bietet sich der NLLLoss an, da er als Vorhersage ein One-Hot Label nimmt und als tatsächliches Label die ID welche Stelle des One-Hot Labels aktiv sein soll

- One-Hot Label:
  Sind 5 Klassen möglich wird die richtige durch eine 1 in einem Array mit 5 Elementen markiert, alle anderen Element sind 0.
  - 0=[1,0,0,0,0]
  - 3=[0,0,0,1,0]

PyTorch – Training & Evaluation

Optimizer

- Der Optimizer kümmert sich um die Anpassung des Netzwerkes auf Basis der ermittelten Gradienten
- PyTorch stellt gängige Algorithmen zur Verfügung, z.B.: Adam, SDG, RMSprop

- Input:
  - Anzupassende Parameter (`model.parameters()`)
  - Lernrate
- Schritt: Um das Netzwerk an die Daten anzupassen, macht er einen Schritt mit der Lernrate als Schrittgröße und den Gradienten als Richtung. (`optimizer.step()`)

- Der Optimizer setzt voraus, dass die Parameter bereits einen Gradienten haben.
  In PyTorch wird das gewährleistet, indem auf dem berechneten Loss-Wert `.backward()` aufgerufen wird

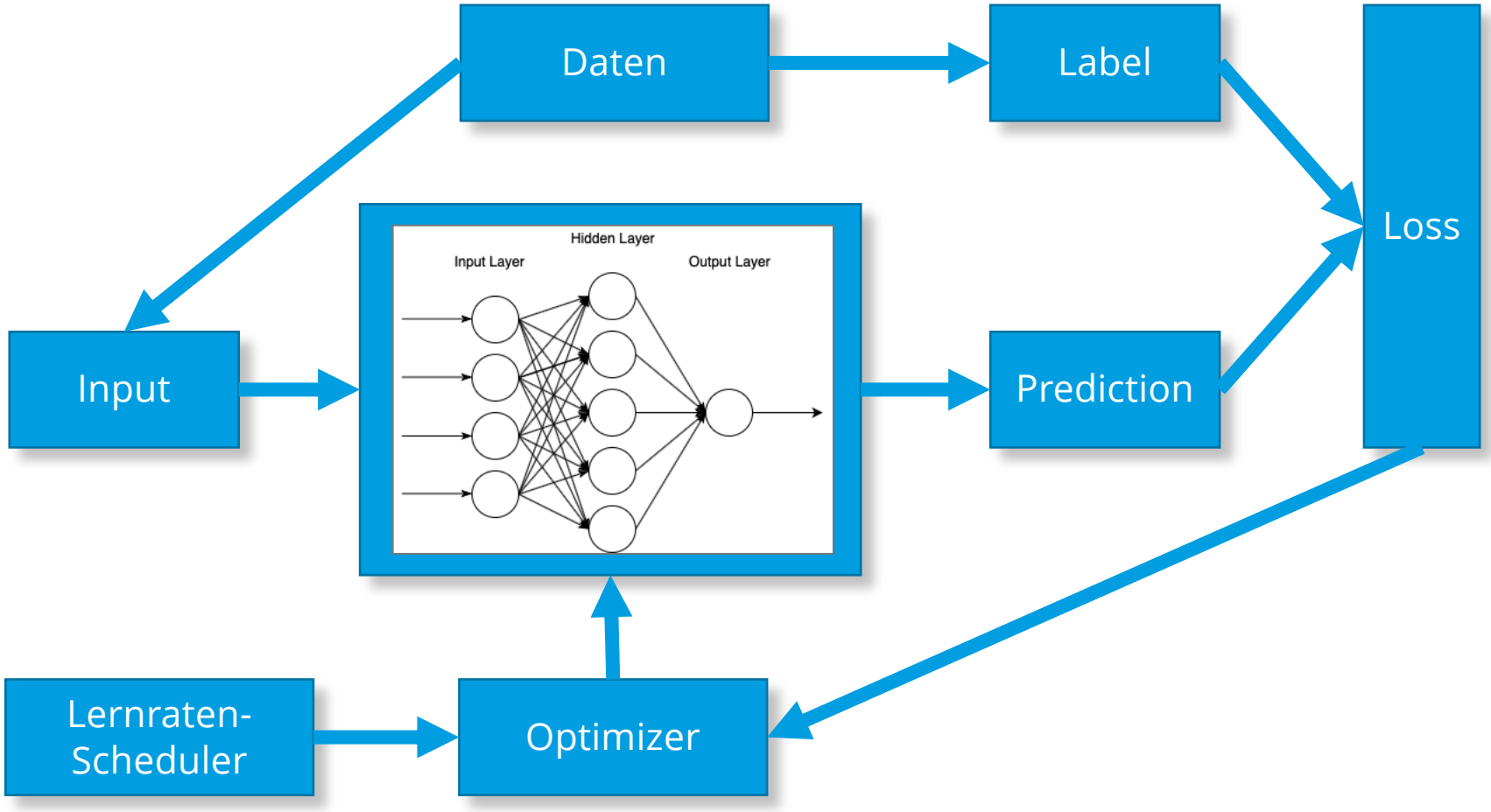
PyTorch – Training & Evaluation

Lernraten-Scheduler

- Über die Zeit wird die Lernrate angepasst
  - Zu beginn muss das Netz noch stark verändert werden, da es noch wenig über das System weis und grobe Fehler macht.
  - Gegen Ende werden nur noch kleine Änderungen vorgenommen, um einzelne Ergebnisse anzupassen

- PyTorch bietet gängige Verfahren zur Lernratenanpassung an, z.B.: StepLR, ReduceLROnPlateau, OneCycleLR

Recap


Daten
Label
Loss
Input
Hidden Layer
Input Layer
Output Layer
Prediction
Lernraten-
Scheduler
Optimizer

Nächste Woche

- F1-Score
- Trainings & Evaluations-Zyklus

- Wir Trainieren unser Netz auf den MNIST Datensatz