Quantitative Textanalyse

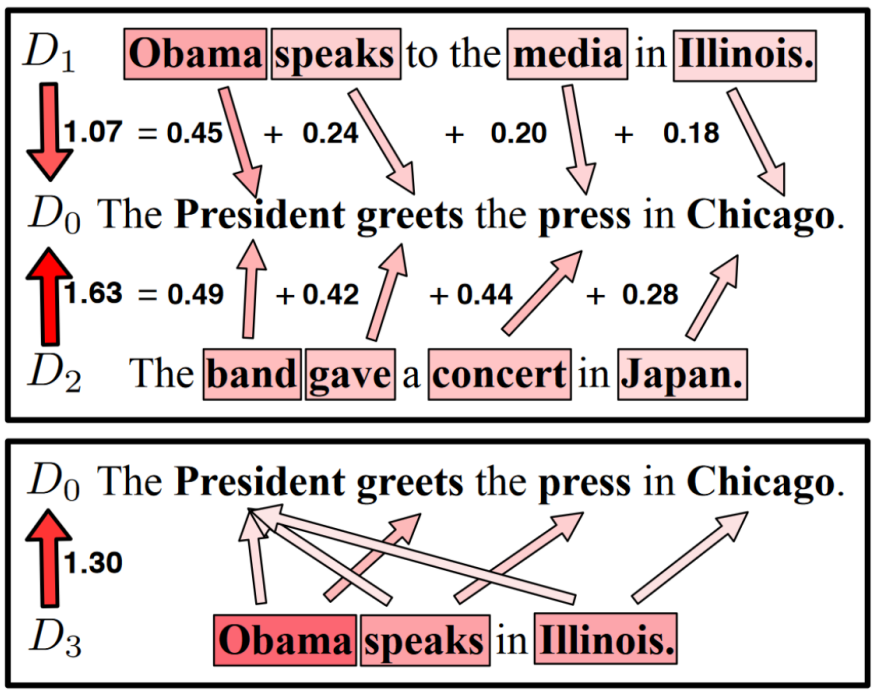
Sitzung 13: Embeddings und Deep Neural Networks

Mirko Wegemann

15. Januar 2025

Ergänzungen zu letzter Woche

Können wir Unterschiede zwischen Dokumenten (D0 und D1) summieren, um den Unterschied zu D3 zu erhalten? **Nein**.



Ergänzungen zu letzter Woche II

Berechnen wir für die Word-Mover-Distance die Entfernung von jedem Wort aus D0 zu jedem Wort aus D1? **Nein**, wir suchen nach Matches zwischen D0 und D1 und berechnen die geringste Entfernung.

Ergänzungen zu letzter Woche III

Enna's Frage zu Lemmatisierung: Müssen wir nach der Lemmatisierung ein neues Embeddings-Objekt erstellen?
Ja, das wäre gut. In unserem Fall sind die Unterschiede marginal. Problematisch kann es aber bei Machine Learning Tasks werden, wenn die Dimensionen der Embeddings-Matrix nicht mit unserem Vokabular übereinstimmen.

Was wir heute machen...

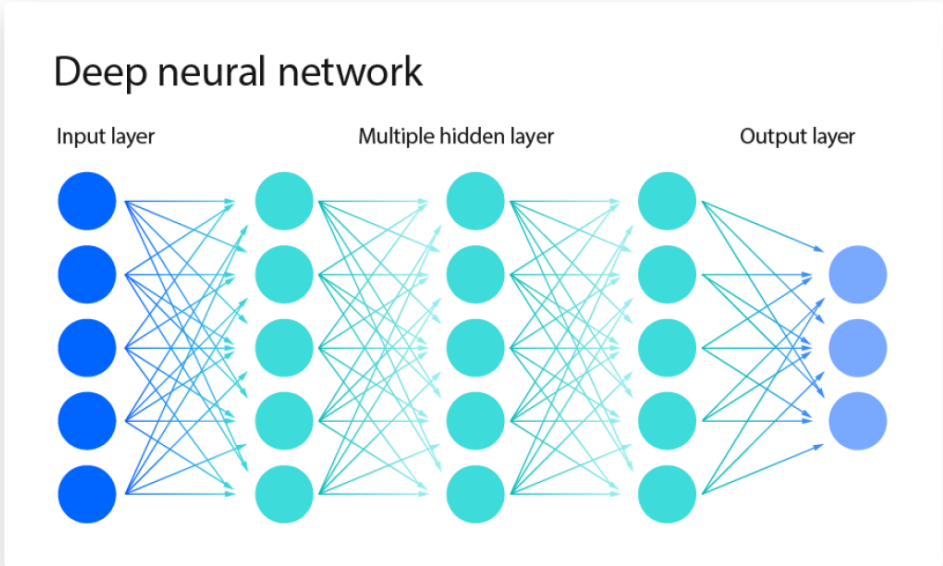
- in der letzten Woche: intrinsische Funktion von Embeddings
- in dieser Woche: Embeddings als Ausgangspunkt für Machine Learning
- bevor wir damit anfangen: Vorstellung des Coding-Tasks von Max

Embeddings und Deep Learning

"Embeddings are numerical representations of real-world objects that machine learning (ML) and artificial intelligence (AI) systems use to understand complex knowledge domains like humans do." Amazon AWS

- wie wir letzte Woche gelernt haben: Embeddings bieten mehr Informationen als Bag-of-Words; zudem ermöglichen sie die Reduktion von Dimensionalität
- daher bilden sie neben ihrer intrinsischen Funktion auch einen Mehrwert für 'Downstream'-Tasks wie der Klassifikation von Texten in Kategorien

Exkurs: Neuronale Netzwerke I



In neuronalen Netzwerken geben wir dem Netzwerk keine genauen Handlungsanweisungen, um unsere Daten interpretieren soll, sondern lassen es eine eigene Funktion lernen, um unsere Daten bestmöglich zu clustern.

Einführung in Deep Learning

Exkurs: Neuronale Netzwerke II

- ein neuronales Netzwerk besteht aus mehreren **Layern** (Schichten), die Input- mit Output-Daten verbinden
- die 'intermediate layers', welche zwischen Input und Output liegen, werden auch '**hidden layers**' genannt

$\rightarrow$ es gibt immer **einen** Input- und **einen** Outputlayer; aber es können mehrere 'hidden layers' vorhanden sein (je mehr, desto komplexer das Model)

Exkurs: Neuronale Netzwerke III

- **cost function**: Abweichung zwischen vorhergesagten und tatsächlichen Werten $\rightarrow$ Fehler
- Aufteilung in **Trainings**- und **Testdaten**: zur Überwachung der Vorhersageleistung [optimalerweise haben wir einen sogenannten 'threefold-split', d.h. eine Aufteilung in Trainings-, Test- und Evaluierungsdaten]
- **Deep Learning**: Neuronales Netzwerk mit mehr als 3 Layers (mehr als einem hidden layer)

Exkurs: Neuronale Netzwerke III

Funktion von neuronalen Netzwerken

- Als Analogie können wir uns die Aktivität eines Gehirns vorstellen: Eine elektrischer Impuls erzeugt ein Signal, das ein weiteres Signal im nächsten Layer aktiviert
- An jeder Schnittstelle wird ein Input empfangen, verarbeitet und als Output weitergegeben – wie bei einem Regressionsmodell
- Wenn der Output einen bestimmten Schwellenwert überschreitet, *aktiviert* er den nächsten Layer und wird zum Input dieses Layers $\rightarrow$ eine Aktivierungsfunktion ermöglicht Nicht-Linearitäten
- Der Fortschritt bzw. die Genauigkeit eines neuronalen Netzwerks wird dabei durch die cost function überwacht

Exkurs: Neuronale Netzwerke IV

Ein praktisches Beispiel

Nehmen wir an, wir wollen folgendes **Outcome** ermitteln: Soll ich surfen gehen oder nicht?[1]

- Drei verschiedene Faktoren beeinflussen meine Entscheidung
  - Sind die Wellen gut?
  - Ist der Strand überfüllt?
  - Gibt es die Gefahr eines Haiangriff?

Jedem Input bzw. jeder Entscheidungsgrundlage weist ein neuronales Netzwerk ein Gewicht bzw. eine Bedeutung zu, das einen Output vorhersagt. Im Vergleich zu Naive Bayes hat also nicht jeder Input die gleiche Bedeutsamkeit für unseren Output.

[1] Quelle: IBM

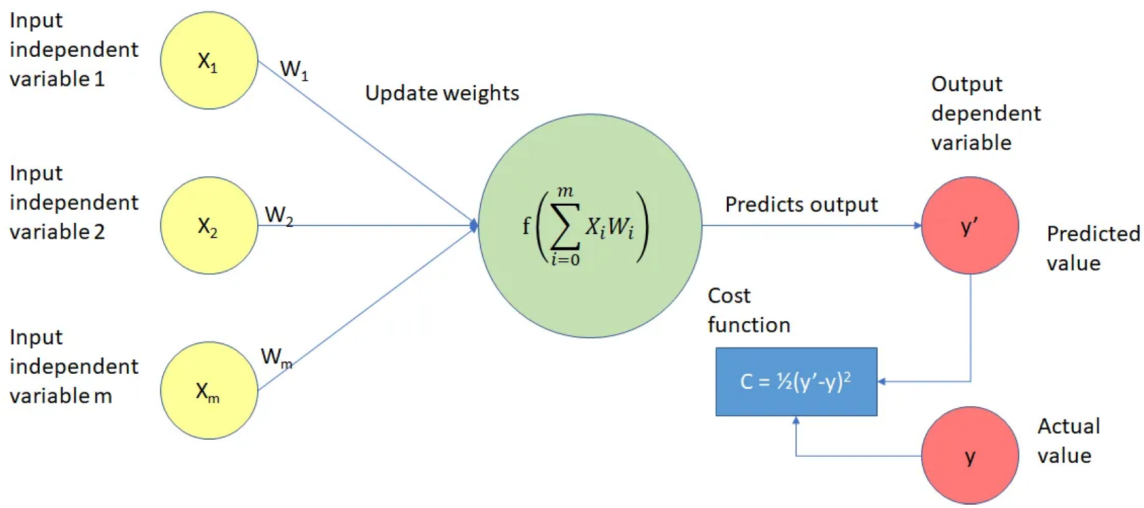
Exkurs: Neuronale Netzwerke IV I

Ein praktisches Beispiel II

- Nach jedem Schritt bzw. jeder Epoche werden die vorhergesagten Outputs (meist Labels/Kategorien) mit den tatsächlichen Werten verglichen (daher die annotierten Testdaten) → Der 'loss' wird berechnet: Wie sehr weichen die Vorhersagen, dass ich surfen gehe, von meinem tatsächlichen Verhalten ab?
- Ziel ist es, den *loss* zu minimieren (und dabei *Overfitting* zu vermeiden, da die Ergebnisse eines neuronalen Netzwerks auch auf andere Daten anwendbar sein sollten)

Exkurs: Neuronale Netzwerke IV II

Ein praktisches Beispiel II



Exkurs: Neuronales Netzwerk VI

Probier dein eigenes neuronales Netzwerk hier aus

Klassifikation in R I

Wir werden *keras3* verwenden, um ein Deep Learning-Modell in R zu definieren.

- **sequentielle** vs. funktionale Modelle
  - In sequentiellen Modellen wird jeder *hidden layer* ausgeführt, bevor der nächste aktiviert wird
  - funktionale Modelle ermöglichen es, Verzweigungen zu erstellen, die gleichzeitig unterschiedliche Ausgaben vorhersagen können

Klassifikation in R II

- **dense** (auch voll verbundene) vs. **convolutional** Layers (für ein besseres Verständnis siehe Mandelbaum and Shalev (2016))
  - *dense layers* stellen einem Modell alle Inputs zur Verfügung, um einen Output zu erzeugen
  - *convolutional layers* verwenden nicht alle Inputs (sondern in unserem Fall nur Wörter, die nahe beieinander liegen) $\rightarrow$ es scannt durch eine Sequenz mit einem festgelegten Kontextfenster n

Hyperparameter I

Wir können verschiedene Parameter anpassen, einige der wichtigsten sind

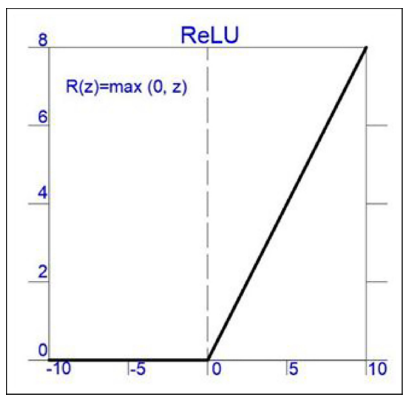
- **hidden layers**: Komplexität eines Modells, mehr Komplexität kann die Präzision erhöhen, aber auch zu Overfitting führen
- **drop-out-rate**: hierbei werden zufällig Informationen gelöscht, um Overfitting zu vermeiden (normalerweise zwischen 0.2-0.5)
- **learning rate**: Wie stark die Gewichtungen nach jedem Schritt aktualisiert werden

Hyperparameter II

- **batches**: die Menge der Daten, die an das neuronale Netzwerk gleichzeitig weitergegeben wird (je größer, desto mehr Speicher wird benötigt; je kleiner, desto weniger präzise)
- **Anzahl der Epochen**: Zyklus eines Lernprozesses (von den Inputs zu den Outputs und zurück zu den Inputs)

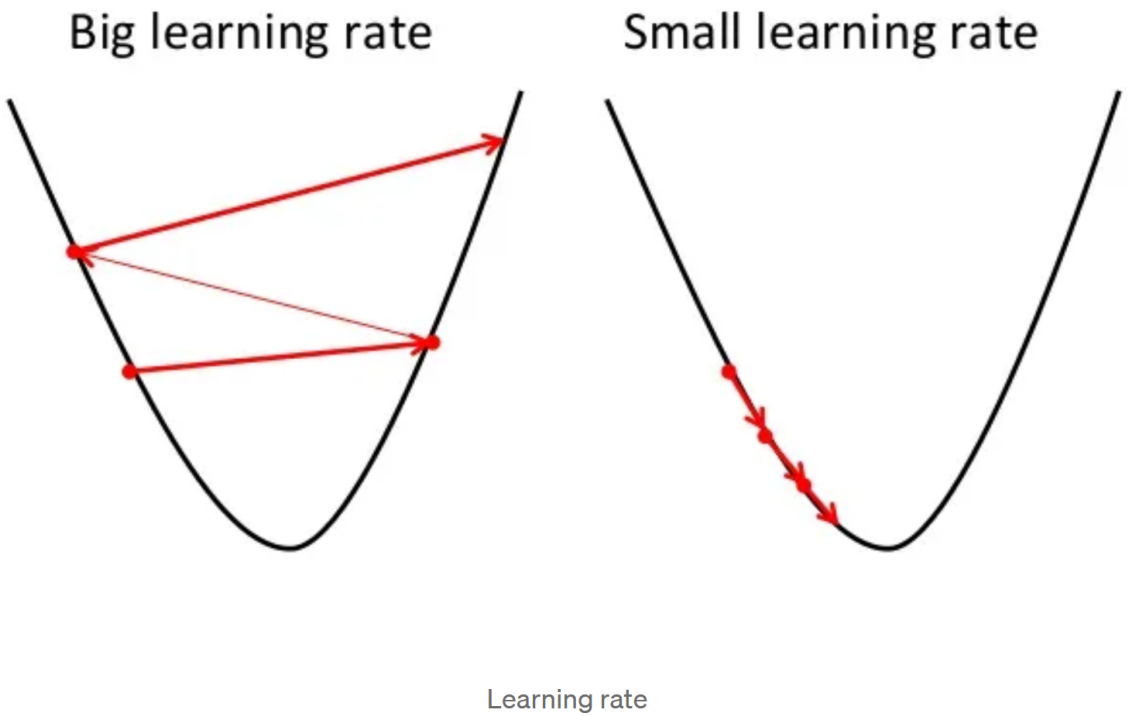
Hyperparameter III

- **activation function**: Erfasst die Nichtlinearität der Daten, verschiedene Funktionen stehen zur Verfügung (*sigmoid* wird oft für binäre Aufgaben verwendet, *softmax* für Mehrklassenvorhersagen) als letzter output layer; *hidden layers* verwenden oftmals *ReLu*



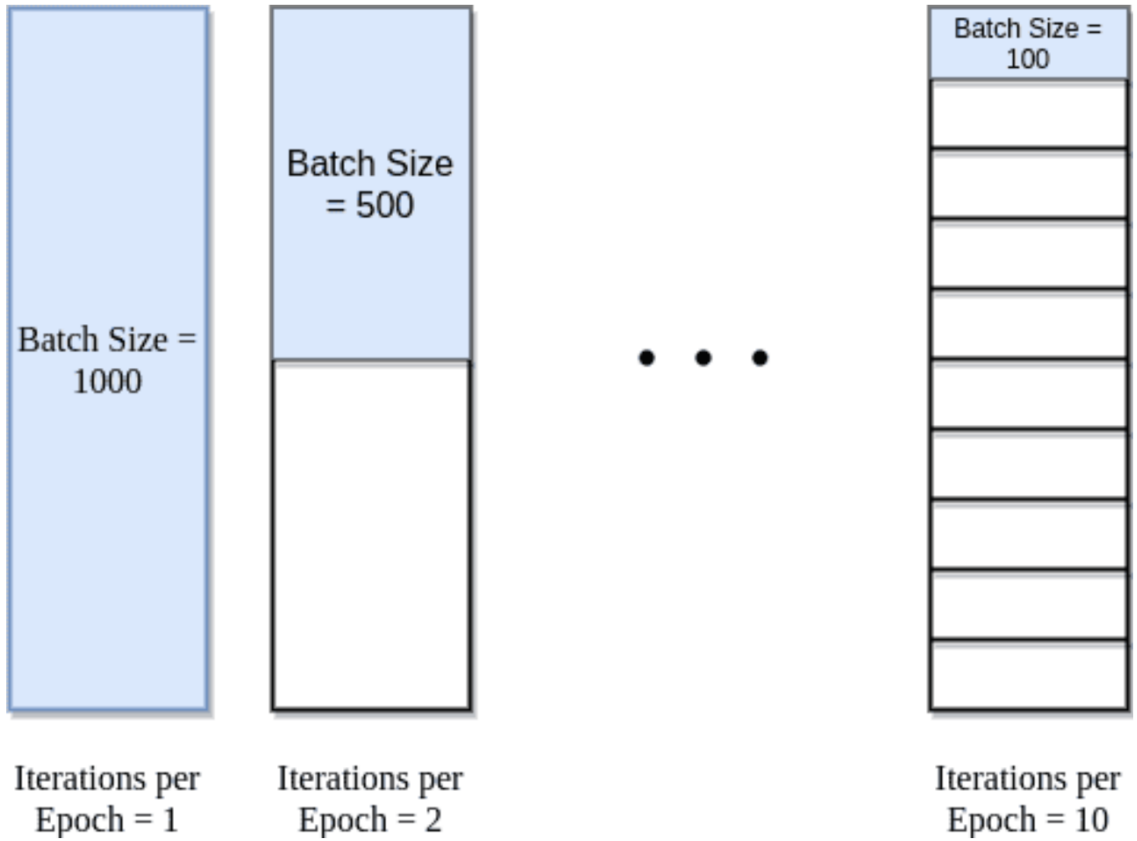
Jedes Mal, wenn ein Schwellenwert ($x > 0$) überschritten wird, wird ein Signal zum nächsten layer des Netzwerks weitergeleitet

Hyperparameter IV
Gradient Descent



Wie unterschiedliche **learning rates** den Fehler beeinflussen

Hyperparameter V



Was **Batch-Größen** im Verhältnis zu **Epochen** bedeuten

Hyperparameter VI

Was **Overfitting** in einem praktischen Beispiel bedeutet

Schritte im Training eines Deep-Learning-Modells

1. Vorbereitung der Eingabedaten (Trainings-/Testaufteilung, Extraktion von Merkmalen und Labels, Umwandlung in numerische Sequenzstruktur)
2. Definition des Modells (Input-, hidden und output layers)
3. Festlegung von Evaluationsmetriken
4. Training des Modells
5. Vorhersage der Validierungsdaten und Evaluierung der Leistung

Vorbereitung

Wie zuvor: Tokenisierung und Erstellung des Vokabulars.
Zusätzlich **Sequenzierung**:

```
1     df_sub$count <- str_count(df_sub$text_prep, "\\w+")
2     max_len <- round(median(df_sub$count,na.rm=T))
3
4     vectorize_layer <- layer_text_vectorization(
5     max_tokens = nrow(vocab),
6     output_sequence_length = max_len
7     )
8     vectorize_layer %>% adapt(df_sub$text_prep)
9     features <- vectorize_layer(df_sub$text_prep)
```

Modell-Definition

Ein Beispiel für ein neuronales Netzwerk mit pre-trained Embeddings als Inputvektor, zwei hidden layers (1 dense, 1 convolutional) und einem Output-Layer.

```
1  model <- keras_model_sequential() %>%
2  layer_embedding(
3  input_dim = dim(embeddings)[1],
4  input_length = max_len,
5  output_dim = dim(embeddings)[2],
6  weights = list(embeddings),
7  trainable = T) %>%
8  layer_conv_1d(filters = 128, kernel_size = 5,
       activation = 'relu') %>%
9  layer_global_max_pooling_1d() %>%
10 layer_dense(units = 128, activation = "relu") %>%
11 layer_dropout(rate = 0.4) %>%
12 layer_dense(units = 1, activation = "sigmoid")
```

Evaluierungsmetriken festlegen

Im nächsten Schritt definieren wir die loss-function, den wir verwenden möchten (für binäre Klassifikationsaufgaben normalerweise *binary_crossentropy*), die Learning Rate und die Metriken, die nach jeder Epoche angezeigt werden

```
1    model %>% compile(
2    loss = "binary_crossentropy",
3    optimizer = optimizer_adam(learning_rate = 0.001),
4    metrics = c('accuracy', metric_precision(),
         metric_recall())
5    )
```

Modell trainieren

Im Schritt der Modellinitialisierung fügen wir zwei weitere Hyperparameter hinzu (*Epochen* und *Batch_Size*)

```
1     history <- model %>% fit(
2     x = x_train,
3     y = y_train,
4     epochs = 20,
5     batch_size = 128,
6     validation_data = list(x_test, y_test),
7     callbacks = list(stop_if_no_improvement)
8     )
```

Vorhersage und Evaluierung

Abschließend sagen wir die Daten vorher und evaluieren (z. B. mit der *confusionMatrix*-Funktion des *caret*-Pakets)

```
1  pred_results <- as.data.frame(predict(model,
       x_test))
2
3  # Klassifiziere Beobachtungen mit
       Wahrscheinlichkeit > 0.5 als 1
4  pred_results$pred <- ifelse(pred_results$V1>=0.5,
       1, 0)
5
6  # Kombiniere Vorhersagen mit den echten annotierten
       Daten
7  pred_results$true <- y_test
8
9  confusionMatrix(as.factor(pred_results$pred),
       as.factor(pred_results$true))
```

...und nun in `R`

Weitere Ressourcen

Hilfreiche Links

- **Tutorial zu Deep Learning**: Practical Deep Learning
- was wir nicht abgedeckt haben, sind Transformer-Models: **Tutorial zu Transformer-Modellen in Colab** Eine praktische Einführung in das Fine-Tuning dieser Modelle von Moritz Laurer
- Übersicht über Language Models von Hugging Face

Was wir heute gelernt haben

- ...wie bereiten wir unsere Daten für komplexere Machine Learning Modelle vor
- ...wie bauen wir unser eigenes Machine Learning Model

Nächste Sitzung

- letzte Sitzung des Seminars
  - Klären offener Fragen
  - Weitere Informationen zum Erwartungshorizont der Hausarbeit
  - Möglichkeit, eure Ideen für die Hausarbeit zu präsentieren

References I

Mandelbaum, A., & Shalev, A. (2016). Word Embeddings and Their Use In Sentence Classification Tasks. https://doi.org/10.48550/arXiv.1610.08229