Recap: Machine Learning

Naive Bayes

- Probabilistic method for classification
- Naive because we ignore feature dependencies
- Prediction model:

$$\arg\max_{c \in C} p(c|f_1, f_2, \ldots, f_n)$$

- Training: Count relative frequencies

Logistic Regression

- Regression method for binary classification
- Output numbers as probabilities
- Prediction model:

$$\frac{1}{1 + e^{-(ax+b)}}$$

- Training: Gradient descent with loss function

Neural Network

- Layered architecture
- Classification type depends on last layer
- Output numbers as probabilities
- Prediction model:

$$L_n(L_{n-1}(L_{\ldots}(L_1(X))))$$

- Training: Backpropagation w/ loss function

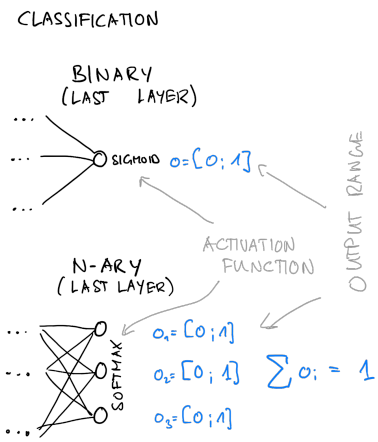
```python
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

# create a random data set with 500 pairs
# of random numbers
x_train = np.random.randn(500,2)

# Target value: Is the sum of two values > 0?
y_train = np.array([1 if sum(x) > 0 else 0
  for x in x_train])

# stop the script for demo purposes
assert False

# setup the model architecture
model = keras.Sequential()
model.add(layers.Input(shape=(2,)))
model.add(layers.Dense(1, activation="sigmoid"))

# compile it
model.compile(loss="binary_crossentropy",
  optimizer="sgd",
  metrics=["accuracy"])

# train it
model.fit(x_train, y_train, epochs=5, batch_size=5)

# create a test data set
x_test = np.random.randn(100,2)
y_test = np.array([1 if sum(x) > 0 else 0
  for x in x_test])
model.evaluate(x=x_test, y=y_test)
```

Hausaufgabe 4

Trainieren Sie ein logistisches Regressionsmodell, um handgeschriebene Ziffern zu erkennen. Die Ziffern wurden handgeschrieben, schwarz/weiß eingescannt und die Bilder dann als 28x28-Matritzen mit Graustufeninformationen bereitgestellt. Es handelt sich nur um Nullen und Einsen, und ist damit eine binäre Klassifikationsaufgabe. Sie finden die Trainings- und Testdaten hier, und hier ein Python-Skript, mit einer Funktion zum Einlesen der Daten. Verwenden Sie die Bibliothek scikit-learn für das eigentliche Training (und schauen Sie sich ruhig ein bisschen um, was die Bibliothek sonst so bereithält).

- Wie hat's geklappt?
- Was kam raus?
- Gab es Schwierigkeiten oder Überraschungen?
- ➔ Hausaufgabe 5: Ziffernerkennung mit neuronalem Netz und einigen selbstgeschriebenen Ziffern

Binary and N-Ary Classification / One-Hot-Encoding



ONE-HOT-ENCODING

- VECTOR TO REPRESENT OUTPUT NUMBER

EXAMPLE

- TRAINING INSTANCE

  CATEGORY 2 (OF 3): $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

- TEST OUTPUT

  CATEGORY 2 : $\begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$

- LOSS CALCULATED BETWEEN $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ AND $\begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$

Last Week

```python
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import LabelBinarizer

# create a random data set with 500 pairs
# of random numbers
x_train = np.random.randn(100,5)

# Target value: What's the maximum of five numbers?
# (0.1, 0.2, -0.2, 0.5, -3)
# -> (4)
y_train = np.array([(np.argmax(x)) for x in x_train])

# one-hot-encoding of target values
lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)

# setup the model architecture
model = keras.Sequential()
model.add(layers.Input(shape=(5,)))
model.add(layers.Dense(20, activation="sigmoid"))
model.add(layers.Dense(5, activation="softmax"))

# compile it
model.compile(loss="categorical_crossentropy",
  optimizer="sgd",
  metrics=["accuracy"])

# train it
model.fit(x_train, y_train, epochs=20, batch_size=1)

# create a test data set
x_test = np.random.randn(100,5)
y_test = np.array([np.argmax(x) for x in x_test])
model.evaluate(x=x_test, y=lb.fit_transform(y_test))
```

- Task: Given five numbers, give us the index of the highest (5-ary classification task)
- 20 epochs, stochastic gradient descent, categorical cross entropy
- 77 % Accuracy

UNIVERSITÄT
ZU KÖLN

Machine Learning: Processing Language with Neural Networks

VL Sprachliche Informationsverarbeitung

Nils Reiter
nils.reiter@uni-koeln.de

December 19, 2023
Winter term 2024/25

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Introduction

A very simple text example

- Task: Given a sentence (with four words), predict wether the sentence is positive or negative
  - E.g., a comment about a book or movie

Introduction

A very simple text example

- Task: Given a sentence (with four words), predict wether the sentence is positive or negative
  - E.g., a comment about a book or movie
- Operationalization
  - Binary classification task
  - Four input features, one for each word
    - Each word gets an index number, which will be the input of the network

demo

s10-example-01.py

Lessons Learned

- Representing words by index numbers alone is not satisfactory
- `{'awesome': 4, 'is': 5, 'terrible': 6, 'bad': 7, 'super': 8}`
  - 'Terrible' and 'bad' are semantically much closer than 'terrible' and 'awesome', but this is not represented
  - Replacing 'bad' with 'terrible' or 'super' is both a change of 1 index position, but has very different meaning

What is Semantics at all?

Man kann für eine große Klasse von Fällen der Benützung des Wortes "Bedeutung" – wenn auch nicht für alle Fälle seiner Benützung – dieses Wort so erklären: Die Bedeutung eines Wortes ist sein Gebrauch in der Sprache. (Wittgenstein, 1953, 20)

What is Semantics at all?

Man kann für eine große Klasse von Fällen der Benützung des Wortes "Bedeutung" – wenn auch nicht für alle Fälle seiner Benützung – dieses Wort so erklären: Die Bedeutung eines Wortes ist sein Gebrauch in der Sprache. (Wittgenstein, 1953, 20)

You shall know a word by the company it keeps (Firth, 1957, 11)

Distributional Semantics

Count vectors

- For each word, we count how often it appears with all other words (within a window of n tokens)
- Results in very long vectors, because all other words
- Many words do not appear with many other words, because of Zipf
  - Many elements are zero

Distributional Semantics

Count vectors

- For each word, we count how often it appears with all other words (within a window of n tokens)
- Results in very long vectors, because all other words
- Many words do not appear with many other words, because of Zipf
  - Many elements are zero

Variants of count vectors

- TF-IDF instead of raw counts
- Mathematical dimensionality reduction

Count Vectors in Our Example

- Words used in similar contexts often get similar vectors
  - E.g., evaluative adjectives like ‘awesome’, ‘super’, ...
  - Antonyms often also get similar vectors

Count Vectors in Our Example

- Words used in similar contexts often get similar vectors
  - E.g., evaluative adjectives like ‘awesome’, ‘super’, ...
  - Antonyms often also get similar vectors
- Recipe
  - Take a large corpus
  - Extract count vectors
  - Insert vectors into our training set

Section 2

Word2Vec

Literature basis

- Two very influential papers by Mikolov et al.
  - Tomáš Mikolov/Kai Chen/Greg Corrado/Jeffrey Dean (2013). "Efficient Estimation of Word Representations in Vector Space". In: *arXiv cs.CL*. URL: https://arxiv.org/pdf/1301.3781.pdf
  - Tomáš Mikolov/Ilya Sutskever/Kai Chen/Greg S Corrado/Jeff Dean (2013). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26 (NIPS 2013)*. Ed. by C. J. C. Burges/L. Bottou/M. Welling/Z. Ghahramani/K. Q. Weinberger. Curran Associates, Inc., pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
- Software package
  - word2vec – `https://github.com/tmikolov/word2vec`
    Originally published on "Google Code"

Basics

- No interpretable dimensions
- Dense vectors: No zeros, and much fewer dimensions than in count vectors

Basics

- No interpretable dimensions
- Dense vectors: No zeros, and much fewer dimensions than in count vectors

Word vectors as a by product

- Recap: Logistic/linear regression and gradient descent
  - Algorithm to fit parameters to a distribution of data points
  - Core ingredient: Loss function
  - Result: Parameter setting θ

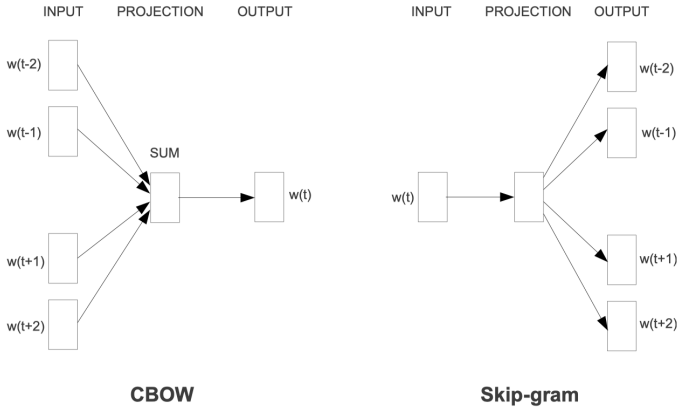
Basics

- No interpretable dimensions
- Dense vectors: No zeros, and much fewer dimensions than in count vectors

Word vectors as a by product

- Recap: Logistic/linear regression and gradient descent
  - Algorithm to fit parameters to a distribution of data points
  - Core ingredient: Loss function
  - Result: Parameter setting θ
- Word2vec
  - Let's use these parameters as word vectors
    - (one parameter vector per word)
  - How to come up with a task that generates these parameters?

Two tasks



Continuous Bag of Words (CBOW)

Context words used to predict one word

Skip-Gram

One word used to predict its context

Skip-gram

- Context: ± 2 words around target word t

```
... dogs, such as a German Shepherd or a Labrador, ...
              c1 c2  t       c3    c4
```

Skip-gram

- Context: ± 2 words around target word t

```
... dogs, such as a German Shepherd or a Labrador, ...
               c1 c2  t        c3    c4
```

- Classifier:
  - Predict for (t, c) wether c are *really* context words for t
  - Probability of $\vec{t}$ and $\vec{c}$ being positive examples: $p(+|\vec{t}, \vec{c})$

Skip-gram

- Context: ± 2 words around target word t

```
... dogs, such as a German Shepherd or a Labrador, ...
               c1 c2  t        c3    c4
```

- Classifier:
  - Predict for (t, c) wether c are *really* context words for t
  - Probability of $\vec{t}$ and $\vec{c}$ being positive examples: $p(+|\vec{t}, \vec{c})$
- Vector similarity $\rightarrow$ probability
  - Similarity of vectors? Dot product 🡻
  - Cosine similarity $\rightarrow$ probability? Logistic function ✔
  - "a word is likely to occur near the target if its embedding is similar to the target embedding"

Jurafsky/Martin (2023, 18 f.)

When are vectors similar?

- Operation that takes two vectors and returns a similarity score
- Linear algebra: dot product
  - A.k.a. scalar product, inner product, Skalarprodukt, Punktprodukt, inneres Produkt

$$\begin{aligned} \vec{a} \cdot \vec{b} &= |\vec{a}||\vec{b}| \cos \sphericalangle(\vec{a}, \vec{b}) \\ &= \sum_{i=1}^{N} a_i b_i \end{aligned}$$

Skip-gram

Notation

t, c: words

$\vec{t}$, $\vec{c}$: vectors for the words

$$p(+|t,c) = \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \sigma(\vec{t}\cdot\vec{c})$$

$$p(-|t,c) = 1 - \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \frac{e^{-\vec{t}\cdot\vec{c}}}{1+e^{-\vec{t}\cdot\vec{c}}} = 1-\sigma(\vec{t}\cdot\vec{c})$$

Skip-gram

Notation

t, c: words

$\vec{t}$, $\vec{c}$: vectors for the words

$$
\begin{aligned}
p(+|t,c) &= \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \sigma(\vec{t}\cdot\vec{c}) \\
p(-|t,c) &= 1 - \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \frac{e^{-\vec{t}\cdot\vec{c}}}{1+e^{-\vec{t}\cdot\vec{c}}} = 1-\sigma(\vec{t}\cdot\vec{c})
\end{aligned}
$$

but the context consists of more than one word!

Skip-gram

Notation
t, c: words
$\vec{t}$, $\vec{c}$: vectors for the words

$$p(+|t,c) = \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \sigma(\vec{t}\cdot\vec{c})$$
$$p(-|t,c) = 1 - \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \frac{e^{-\vec{t}\cdot\vec{c}}}{1+e^{-\vec{t}\cdot\vec{c}}} = 1-\sigma(\vec{t}\cdot\vec{c})$$

but the context consists of more than one word!
Assumption: They are independent, allowing multiplication

Skip-gram

Notation
t, c: words
$\vec{t}$, $\vec{c}$: vectors for the words

$$
\begin{aligned}
p(+|t,c) &= \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \sigma(\vec{t}\cdot\vec{c}) \\
p(-|t,c) &= 1 - \frac{1}{1+e^{-\vec{t}\cdot\vec{c}}} = \frac{e^{-\vec{t}\cdot\vec{c}}}{1+e^{-\vec{t}\cdot\vec{c}}} = 1-\sigma(\vec{t}\cdot\vec{c})
\end{aligned}
$$

but the context consists of more than one word!
Assumption: They are independent, allowing multiplication

$$
\begin{aligned}
p(+|t,c_{1:k}) &= \prod_{i=1}^{k} \frac{1}{1+e^{-\vec{t}\cdot\vec{c}_i}} \\
\log p(+|t,c_{1:k}) &= \sum_{i=1}^{k} \log \frac{1}{1+e^{-\vec{t}\cdot\vec{c}_i}}
\end{aligned}
$$

Skip-gram

- So far, we have assumed that we have vector $\vec{t}$ for word t, but where do they come from?
- Basic gradient descent: We start randomly, and iteratively improve

Skip-gram

Negative sampling

- Negative examples
  - Training a classifier needs negative examples, i.e., words that are not in the context of each other

Skip-gram

Negative sampling

- Negative examples
  - Training a classifier needs negative examples, i.e., words that are not in the context of each other
- Negative sampling
  - For every positive tuple (t, c), we add k negative tuples
  - Negative tuple (t, c_n), with c_n randomly selected (and $t \neq c_n$)

Skip-gram

Negative sampling

- Negative examples
  - Training a classifier needs negative examples, i.e., words that are not in the context of each other
- Negative sampling
  - For every positive tuple (t, c), we add k negative tuples
  - Negative tuple (t, c_n), with c_n randomly selected (and $t \neq c_n$)
- New 'parameter' k on this slide
  - Different status than θ (the parameters we want to learn)
  - Therefore: Hyperparameters

Word2Vec

Loss

- We also need a loss function
- Idea:
  - Maximize
    - $p(+|t, c)$ (positive samples), and
    - $p(-|t, c_n)$ (negative samples)

Word2Vec

Loss

- We also need a loss function
- Idea:
  - Maximize
    - $p(+|t, c)$ (positive samples), and
    - $p(-|t, c_n)$ (negative samples)

$$L(\theta) = \sum_{(t,c)} \log p(+|t, c) + \sum_{(t,c_n)} \log p(-|t, c_n)$$

Word2Vec

Loss

- We also need a loss function
- Idea:
  - Maximize
    - $p(+|t, c)$ (positive samples), and
    - $p(-|t, c_n)$ (negative samples)

$$L(\theta) = \sum_{(t,c)} \log p(+|t, c) + \sum_{(t,c_n)} \log p(-|t, c_n)$$

θ: Concatenation of all $\vec{t}$, $\vec{c}$, $\vec{c}_n$

Remarks and observations

- Each word is used twice, with different roles
  - As target word (for predicting its context)
  - As context word (to be predicted from another target word)
  - Different options: Only use one embedding, combine them by addition or concatenation

Section 3

Embeddings and Neural Networks

Two Options

- Embedding: Each token is replaced by a vector of numbers

Two Options

- Embedding: Each token is replaced by a vector of numbers
- Option 1
  - Download pre-trained embeddings (e.g., via word2vec)
  - Replace them during preprocessing
  - Benefit from large training set

Two Options

- Embedding: Each token is replaced by a vector of numbers
- Option 1
  - Download pre-trained embeddings (e.g., via word2vec)
  - Replace them during preprocessing
  - Benefit from large training set
- Option 2
  - Train your own embeddings in your neural network
  - In the end, it's just more parameters to learn, and we know how to do that
  - Keras: `keras.layers.Embedding`

demo

s10-example-02.py, s10-example-03.py

Section 4

Summary

Summary

Represent text data in neural networks

- Map words to indices
- Embeddings
  - Way to represent input data
  - Word2Vec: Concrete method to calculate/train embeddings
  - Well suited as input for neural networks
  - Pre-trained embeddings
    - Easy to use
    - Trained on very large corpora
    - Allow to incorporate some kind of knowledge into our own models that we don't have to annotate