Recap: Embeddings

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

Machine Learning 5: Overfitting & Sequence Labeling

VL Sprachliche Informationsverarbeitung

Nils Reiter
nils.reiter@uni-koeln.de

December 22, 2022
Winter term 2022/23

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Overfitting

Introduction

- 'Fitting': Train a model on data (= "fit" it to the data)
  - Underfitting: The model is not well fitted to the data, i.e., accuracy is low
  - Overfitting: The model is fitted too well to the data, i.e., accuracy is high

Introduction

- 'Fitting': Train a model on data (= "fit" it to the data)
  - Underfitting: The model is not well fitted to the data, i.e., accuracy is low
  - Overfitting: The model is fitted too well to the data, i.e., accuracy is high

Why is overfitting a problem?

Introduction

- 'Fitting': Train a model on data (= "fit" it to the data)
  - Underfitting: The model is not well fitted to the data, i.e., accuracy is low
  - Overfitting: The model is fitted too well to the data, i.e., accuracy is high

Why is overfitting a problem?

- We want to the model to behave well "in the wild"
- It needs to generalize from training data
- If it is overfitted, it works very well on training data, and very badly on test data

Intuition

$\simeq$ Learning by heart

Example

- Learning by heart gets you through the test
  - I.e., systems achieve high performance

Intuition

≃ Learning by heart

Example

- Learning by heart gets you through the test
  - I.e., systems achieve high performance
- You are unable to apply your knowledge to situations not exactly as in the test
  - I.e., system performance is lower in the wild

Intuition

$\simeq$ Learning by heart

Example

- Learning by heart gets you through the test
  - I.e., systems achieve high performance
- You are unable to apply your knowledge to situations not exactly as in the test
  - I.e., system performance is lower in the wild

Real-World Examples

This is an excellent collection of examples for overfitting: https://stats.stackexchange.com/questions/128616/whats-a-real-world-example-of-overfitting

Overfitting and Neural Networks

Classical machine learning

- Feature selection can avoid relying on irrelevant features
- ⚠ Only one source for overfitting

Overfitting and Neural Networks

Classical machine learning

- Feature selection can avoid relying on irrelevant features
- ⚠ Only one source for overfitting

Neural networks are overfitting machines

- Layered architecture $\Rightarrow$ Any relation between x and y can be learned
  - including a fixed set of if/else rules

Techniques against overfitting

- Regularization
- Dropout

Section 2

Regularization

Intuition

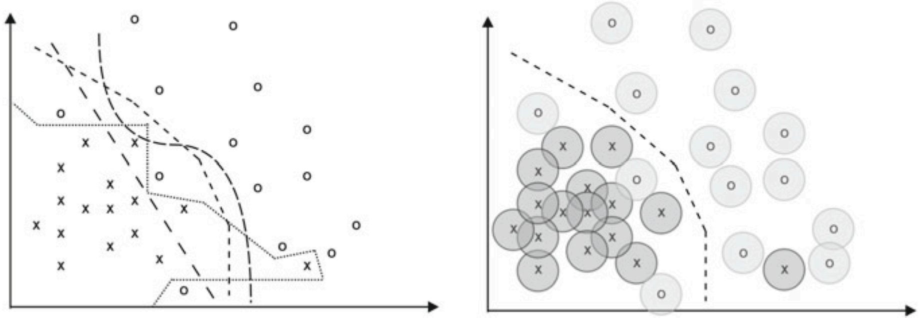


Figure: Visual representation of regularization results (Skansi, 2018, 108)

Formalization

- Formally, regularization is a parameter added to the loss

$$J(\vec{w}) = J_{\mathsf{original}}(\vec{w}) + R$$

L^2-Regularization

L^2-Norm (a. k. a. Euclidean norm) Tikhonov (1963)

- Given a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$,
  its L^2 norm is $L^2(\vec{x}) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = ||\vec{x}||_2$

L^2-Regularization

L^2-Norm (a. k. a. Euclidean norm) Tikhonov (1963)

- Given a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$,
  its L^2 norm is $L^2(\vec{x}) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = ||\vec{x}||_2$
- In practice, we drop the square root and calculate L^2 norm of the weight vector during training:

$$(||\vec{w}||_2)^2 = \sum_{i=0}^{n} w_i^2$$

L^2-Regularization

L^2-Norm (a. k. a. Euclidean norm) Tikhonov (1963)

- Given a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$,
  its L^2 norm is $L^2(\vec{x}) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = ||\vec{x}||_2$
- In practice, we drop the square root and calculate L^2 norm of the weight vector during training:

$$(||\vec{w}||_2)^2 = \sum_{i=0}^{n} w_i^2$$

- Regularization rate λ: Factor that expresses how much we want (another hyperparameter)

$$J(\vec{w}) = J_{\text{original}}(\vec{w}) + \frac{\lambda}{n}||w||_2^2 \qquad \text{with } n \text{ for the batch size}$$

L_2-Regularization

- What does it do?

L_2-Regularization

- What does it do?
  - If weights $\vec{w}$ are large: Loss is increased more
  - Large weights are only considered if the increased loss is “worth it”, i.e., if it is counterbalanced by a real error reduction
  - Small weights are preferred

L^1-Regularization (Tibshirani, 1996)

- Absolute values instead of squares

$$L^1(\vec{x}) = \sum_{i=0}^{n} |x_i|$$

L^1-Regularization (Tibshirani, 1996)

- Absolute values instead of squares

$$L^1(\vec{x}) = \sum_{i=0}^{n} |x_i|$$

L^1 or L^2?

- Skansi (2018):
  - In most cases: L^2 is better
  - Use L^1 if data is very noisy or sparse

Implementation

- In Keras, most layers support additional arguments for regularization:
  - `kernel_regularizer`, `bias_regularizer`, `activity_regularizer`
    - Applied to weights, constant term, neuron output (= result of activation function)
    - Docs: `https://keras.io/api/layers/regularizers/`

Implementation

- In Keras, most layers support additional arguments for regularization:
  - `kernel_regularizer`, `bias_regularizer`, `activity_regularizer`
    - Applied to weights, constant term, neuron output (= result of activation function)
    - Docs: `https://keras.io/api/layers/regularizers/`
  - Argument value: Regularization function with parameter(s)
    - Layer-specific

Implementation

- In Keras, most layers support additional arguments for regularization:
  - `kernel_regularizer`, `bias_regularizer`, `activity_regularizer`
    - Applied to weights, constant term, neuron output (= result of activation function)
    - Docs: `https://keras.io/api/layers/regularizers/`
  - Argument value: Regularization function with parameter(s)
    - Layer-specific

```
ffnn.add(layers.Dense(5,
  activation="sigmoid",
  activity_regularizer=regularizers.l2(0.2)))
```

Section 3

Dropout

Dropout

- Regularization: Numerically combatting overfitting
- Dropout: Structurally combatting overfitting Hinton et al. (2012)

Dropout

- Regularization: Numerically combatting overfitting
- Dropout: Structurally combatting overfitting — Hinton et al. (2012)
  - A new hyperparameter $\pi = [0; 1]$
  - In each epoch, every weight is set to zero with a probability of π

Dropout

Example

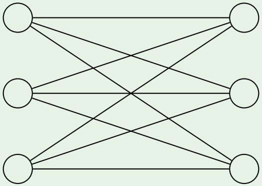


Figure: Dropout $\pi = 0.5$, visualized

Dropout

Example

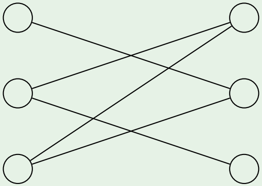


Figure: Dropout $\pi = 0.5$, visualized, Epoch 0

Dropout

Example

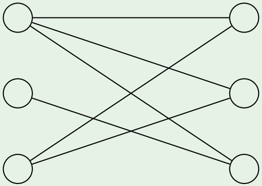


Figure: Dropout $\pi = 0.5$, visualized, Epoch 1

Dropout

Example

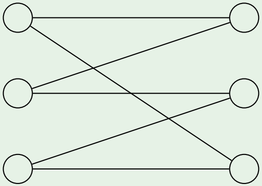


Figure: Dropout $\pi = 0.5$, visualized, Epoch 2

Dropout
Implementation

- Why?
  - Dropout forces the network to learn redundancies

Dropout

Implementation

- Why?
  - Dropout forces the network to learn redundancies
- Implementation
  - In Keras, dropout is realized as additional layer
  - Applies to the layer before the dropout layer

```
model.add(layers.Dense(10)) # no edges dropped
model.add(layers.Dense(20)) # edges are dropped here
model.add(layers.Dropout(0.5))
```

Section 4

Sequence Labeling

Motivation

- Language works sequentially
  - Word meaning depends on context

Motivation

- Language works sequentially
  - Word meaning depends on context
- Feedforward neural networks
  - One instance at a time
  - E.g., one sentence with four tokens ➔ positive/negative

Motivation

- Language works sequentially
  - Word meaning depends on context
- Feedforward neural networks
  - One instance at a time
  - E.g., one sentence with four tokens ➔ positive/negative
- Conceptually not adequate for natural language
- Length of influencing context varies

Motivation

- Language works sequentially
  - Word meaning depends on context
- Feedforward neural networks
  - One instance at a time
  - E.g., one sentence with four tokens ➔ positive/negative
- Conceptually not adequate for natural language
- Length of influencing context varies
- Recurrent neural networks are one solution to this problem

Sequence Labeling

- So far: Classification
- Sequence labeling
  - Special case of classification
  - Instances are organized sequentially and not independent of each other
    - I.e.: The prediction for one class influences the next

Sequence Labeling

- So far: Classification
- Sequence labeling
  - Special case of classification
  - Instances are organized sequentially and not independent of each other
    - I.e.: The prediction for one class influences the next

Example (Part of speech tagging)

"the dog barks" $\rightarrow$ "DET NN VBZ"

BIO Scheme

- Named entity recognition is complicated
  - Not every token is part of a named entity (NE)
  - Many named entities span multiple tokens
  - We distinguish NEs based on the ontological type of the referent
    - PERson, ORGanization, LOCation, …

BIO Scheme

- Named entity recognition is complicated
  - Not every token is part of a named entity (NE)
  - Many named entities span multiple tokens
  - We distinguish NEs based on the ontological type of the referent
    - PERson, ORGanization, LOCation, ...
- BIO scheme
  - How to represent NE annotations token-wise
  - Each token gets a label
    - B: Beginning of a NE
    - I: Inside of a NE
    - O: Outside of a NE (the majority of tokens)

BIO Scheme

- Named entity recognition is complicated
  - Not every token is part of a named entity (NE)
  - Many named entities span multiple tokens
  - We distinguish NEs based on the ontological type of the referent
    - PERson, ORGanization, LOCation, …
- BIO scheme
  - How to represent NE annotations token-wise
  - Each token gets a label
    - B: Beginning of a NE
    - I: Inside of a NE
    - O: Outside of a NE (the majority of tokens)
- Why B: Marking the beginning allows to recognize multiple multi-word NEs in direct sequence
  - "…hat Peter Paulus Maria Müller geküsst" → "O B-PER I-PER B-PER I-PER O"

Towards Recurrent Neural Networks

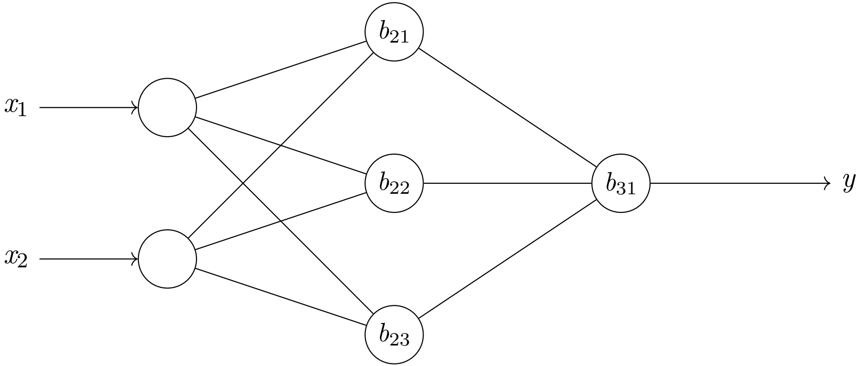


Figure: A feedforward neural network with 1 hidden layer (same picture as before)

Towards Recurrent Neural Networks

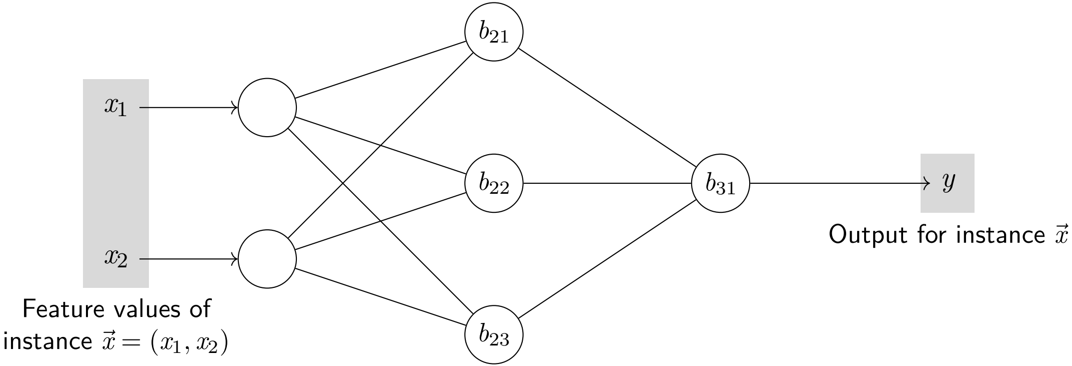


Figure: A feedforward neural network with 1 hidden layer (same picture as before)

Towards Recurrent Neural Networks

To work with sequences, we need to include the sequence into the model

Notation

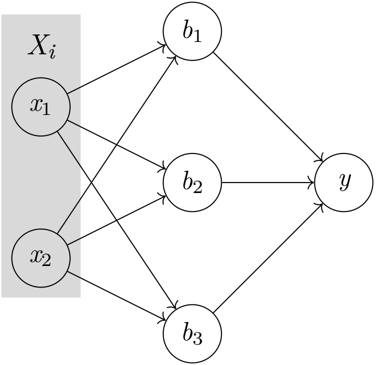
$X = (\vec{X}_1, \vec{X}_2, \dots)$ The input data set containing a sequence of instances (e.g., a sequence of words)

$\vec{X}_i = (x_1, x_2, \dots)$ One instance with feature values (e.g., embedding dimensions)

Y_i Output for instance X_i

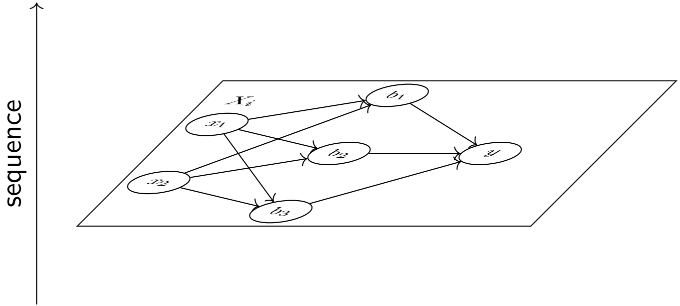
Recurrent Neural Networks

Example



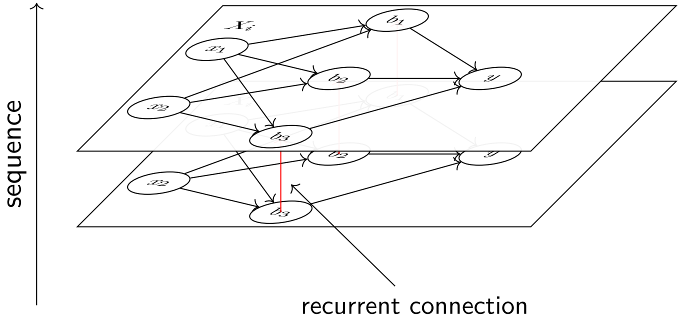
Recurrent Neural Networks

Example



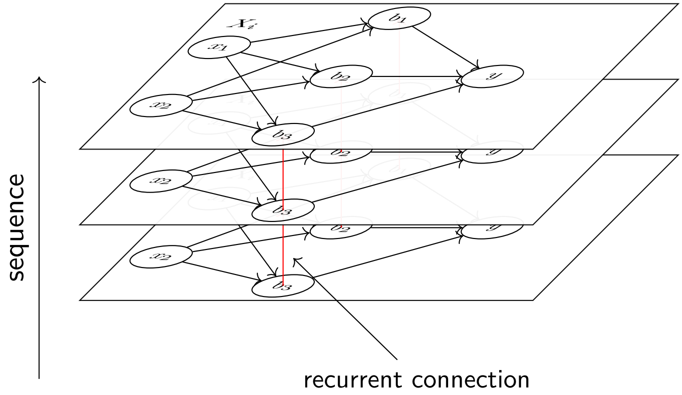
Recurrent Neural Networks

Example



Recurrent Neural Networks

Example



Recurrent Neural Networks

- FFNN, CNN: Weights between neurons
- RNN
  - Weights between neurons
  - Weight(s) for recurrent connections

Recurrent Neural Networks

- FFNN, CNN: Weights between neurons
- RNN
  - Weights between neurons
  - Weight(s) for recurrent connections

Input shape

RNN layers need 2D input:

- Length of input sequences (if needed, padded)
- Number of features (dimensions)
  - (this is where embeddings would go)

Demo

- Simple task: Learn to count distances
  - Given a sequence of 1s and 0s, predict a 1 two steps after an input-1
  - E.g.: “010010001” becomes “000100100”
  - Model has to learn to count the distance
  - Training data can easily be generated

Demo

- Simple task: Learn to count distances
  - Given a sequence of 1s and 0s, predict a 1 two steps after an input-1
  - E.g.: “010010001” becomes “000100100”
  - Model has to learn to count the distance
  - Training data can easily be generated

demo

Implementation in keras

- `tf.keras.layers.SimpleRNN`
  - Documentation: `https://keras.io/api/layers/recurrent_layers/simple_rnn/`
    Selected parameters:
  - `recurrent_dropout=0.0` Dropout for recurrent links
  - `return_sequences=False` Wether to return the entire sequence or just the last element

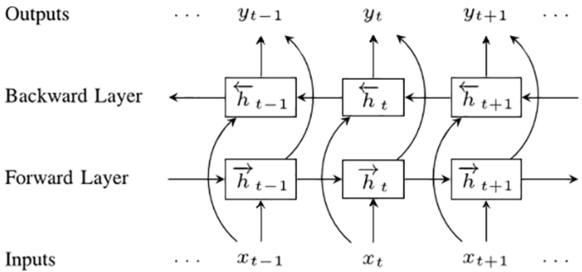
```
model.add(layers.SimpleRNN(...))
```

Directions

- In a regular RNN, the sequence is processed in one direction
- Simple extension: two recurrent layers for both directions

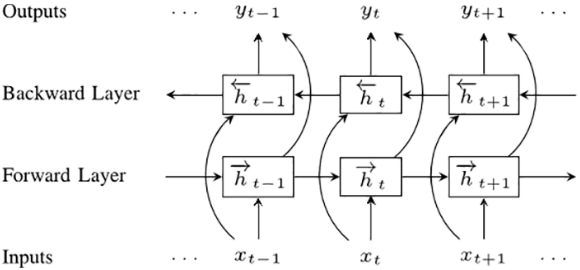
Directions

- In a regular RNN, the sequence is processed in one direction
- Simple extension: two recurrent layers for both directions



Directions

- In a regular RNN, the sequence is processed in one direction
- Simple extension: two recurrent layers for both directions




```
model.add(layers.Bidirectional(layers.SimpleRNN(...)))
```

Section 5

Summary

Summary

Overfitting

- Bla