# Text as Data

## Session 11: Embeddings und Deep Neural Networks

Mirko Wegemann

15 July 2026

# Additions to last week

For the word-mover distance, do we calculate the distance from each word from D0 to each word from D1?

**No**, we look for matches between D0 and D1 and calculate the shortest distance.

# What we're doing today...

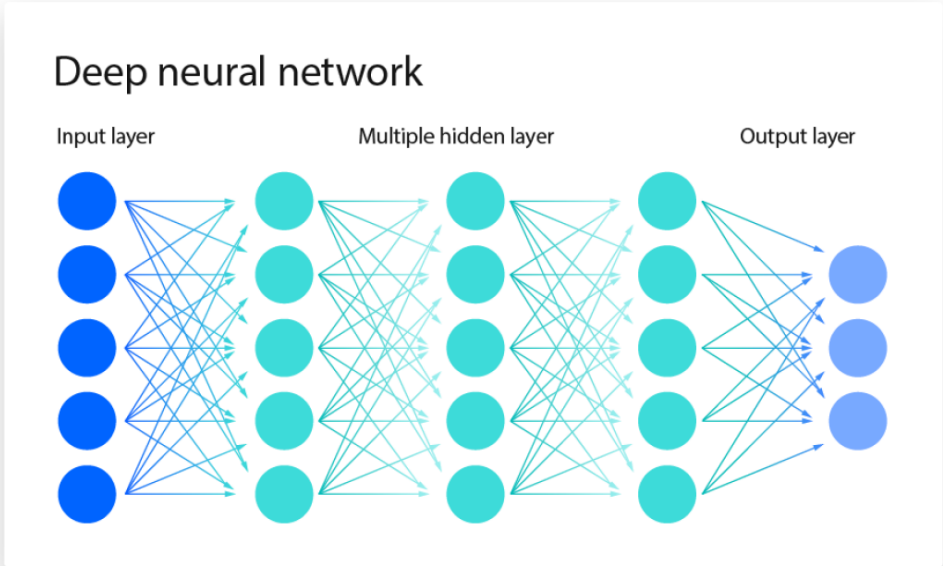
- Course evaluation
- ...descriptive function of embeddings
- ...and their value as a starting point for machine learning
- ...maybe brief discussion of embeddings

# Embeddings and 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

- Embeddings provide more information than bag-of-words; they also enable the reduction of dimensionality
- They also provide added value for 'downstream' tasks such as the classification of texts into categories

# Neural Networks I




In neural networks, we do not give the network precise instructions on how to interpret our data, but let the model learn its own function in order to cluster our data in the best possible way.
Introduction to Deep Learning

# Neural Networks II

- A neural network consists of several **layers** (layers) that connect input and output data
- the 'intermediate layers', that lie between input and output, are also called '**hidden layers**'

$\rightarrow$ there are always **one** input and **one** output layer; but there can be several 'hidden layers' (the more hidden layers, the more complex the model)

# Neural Networks III

- **cost function**: Variance between predicted and actual values $\rightarrow$ Errors
- Split into **training** and **test data**: to monitor prediction performance [optimally, we have a so-called 'threefold split', i.e. a split into training, test and evaluation data]
- **Deep Learning**: Neural network with more than 3 layers (more than one hidden layer)

# Neural Networks III

## Function of neural networks

- As an analogy, we can think of the activity of a brain: an electrical impulse generates a signal that activates another signal in the next layer
- At each interface, an input is received, processed, and passed on as output – like a regression model
- If the output exceeds a certain threshold, *activates* the next layer and becomes the input of that layer $\rightarrow$ an activation function allows non-linear relationships to be taken into account
- The progress or accuracy of a neural network is monitored by the cost function

# Neural Networks IV

## A practical example

Let's say we want to determine the following **Outcome**: Should I go surfing or not?[1]

- Three different factors influence my decision
  1. Are the waves good?
  2. Is the beach crowded?
  3. Is there a risk of a shark attack?

A neural network assigns a weight or meaning to each input or basis for decision-making, which predicts an output.

[1] Source: IBM

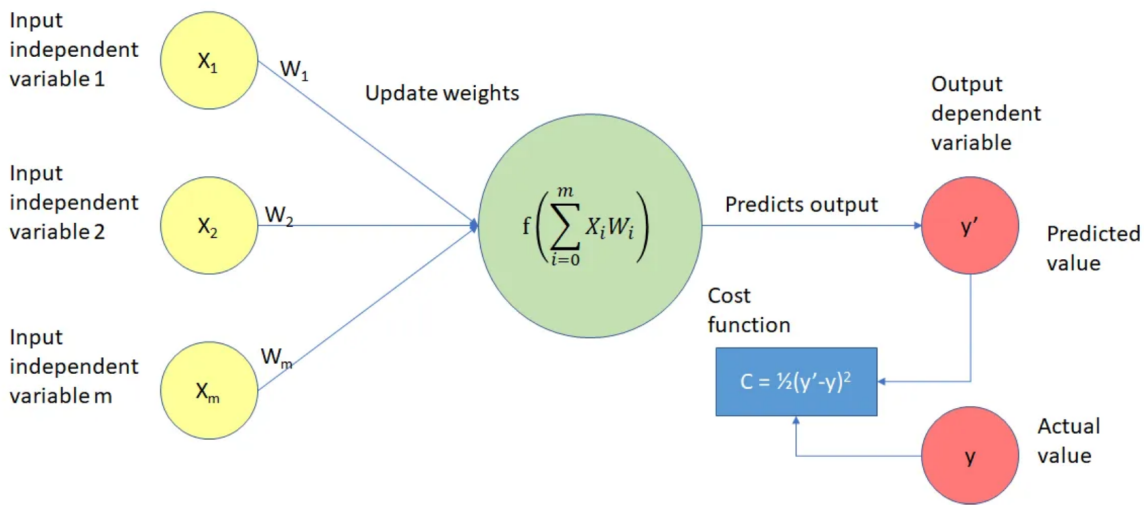
# Neural Networks IV I

## A Practical Example II

- After each step or epoch, the predicted outputs (mostly labels/categories) are compared with the actual values (hence the annotated test data) $\rightarrow$ The 'loss' is calculated: How much do the predictions that I will go surfing differ from my actual behavior?
- The goal is to minimize *loss* (while avoiding *overfitting*, since the results of a neural network should be applicable to other data as well)

# Neural Networks IV II

## A Practical Example II



# Neural Network VI

Try your own neural network here

# classification in R I

We're going to use *keras3* to define a deep learning model in R.

- **sequential** vs. functional models
  - In sequential models, each *hidden layer* is executed before the next one is activated
  - functional models make it possible to create branches that can predict different outputs at the same time

# classification in R II

- **dense** (also fully connected) vs. **convolutional** layers (for a better understanding, see **Mandelbaum.2016**)
  - *dense layers* provide a model with all the inputs to produce an output
  - *convolutional layers* don't use all inputs (but in our case only words that are close to each other) $\rightarrow$ it scans through a sequence with a set context window $n$

# hyperparameters I

We can adjust various parameters, some of the most important ones are

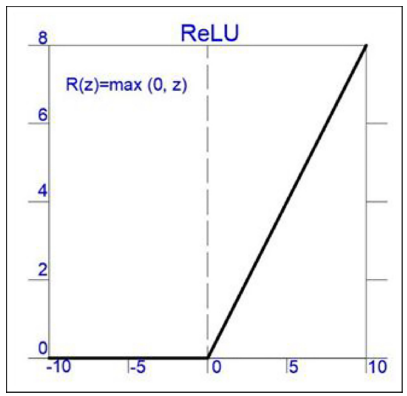
- **hidden layers**: Complexity of a model, more complexity can increase precision, but also lead to overfitting
- **drop-out-rate**: this randomly deletes information to avoid overfitting (usually between 0.2-0.5)
- **learning rate**: How much the weights are updated after each step

# hyperparameters II

- **batches**: the amount of data that is passed to the neural network at the same time (the larger, the more memory is needed; the smaller, the less precise)
- **number of epochs**: cycle of a learning process (from inputs to outputs and back to inputs)

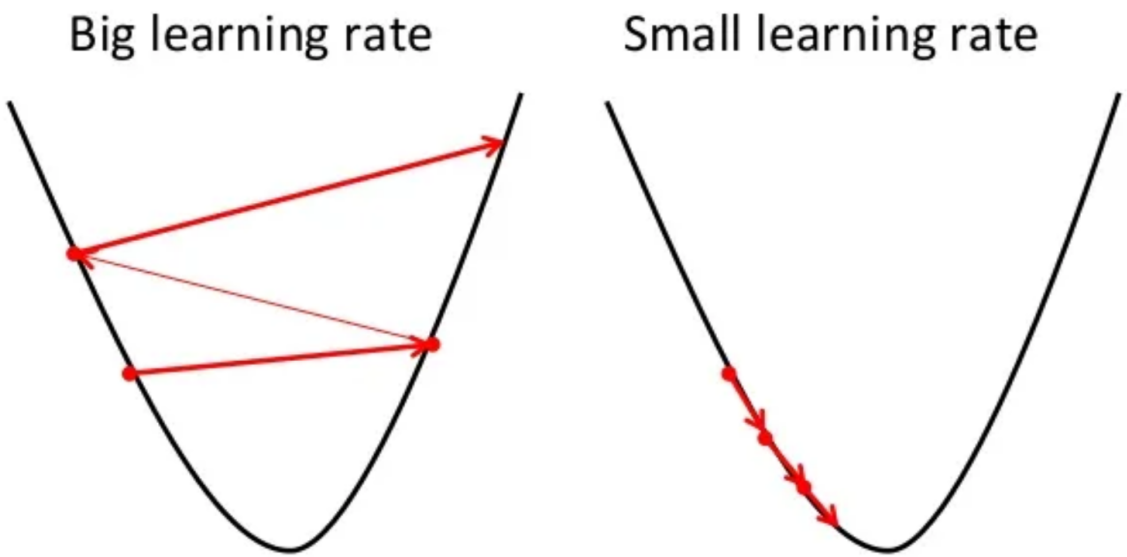
# hyperparameters III

- **activation function**: Captures the nonlinearity of the data, various functions are available (*sigmoid* is often used for binary tasks, *softmax* for multiclass predictions) as the last output layer; *hidden layers* often use *ReLu*



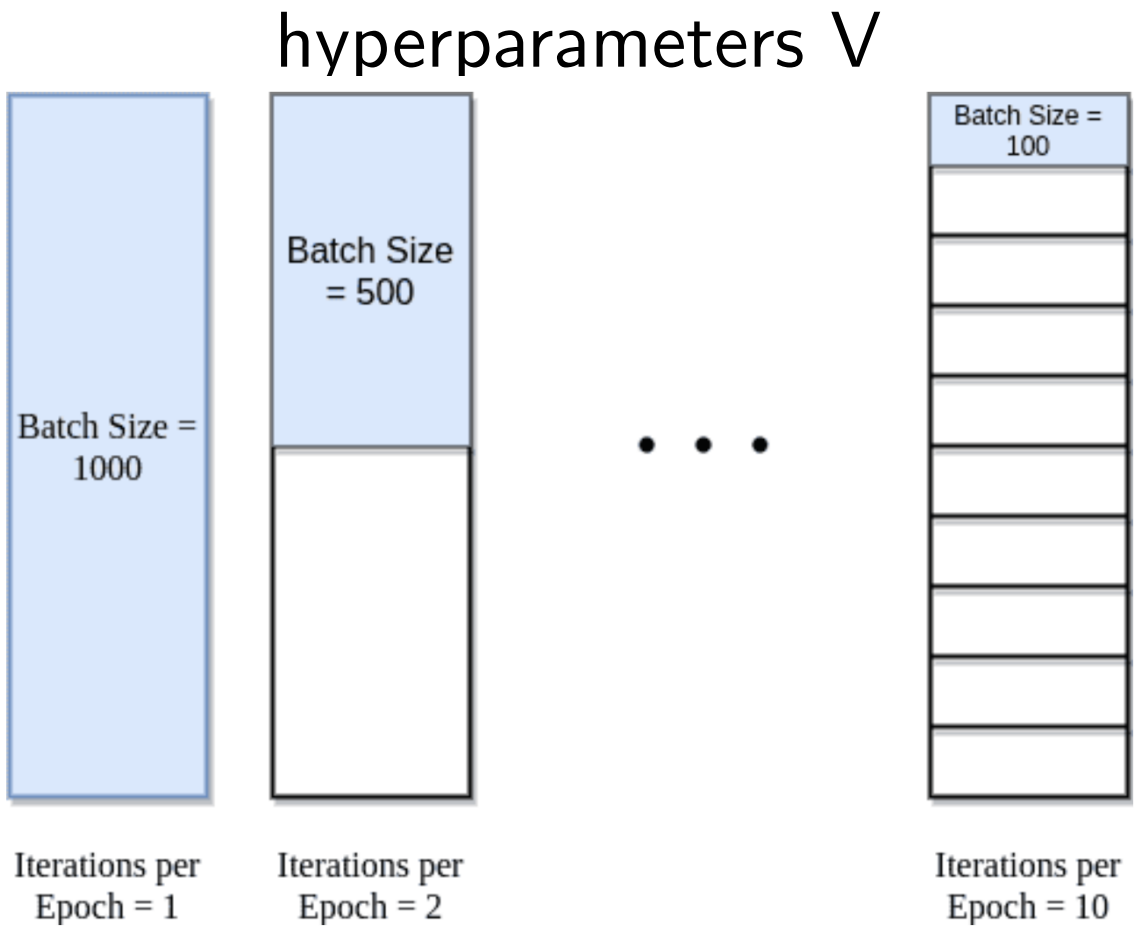

Each time a threshold value ($x > 0$), a signal is forwarded to the next layer of the network

# hyperparameters IV
## Gradient Descent




Learning rate

How different **learning rates** affect the error




What **batch sizes** mean in relation to **epochs**

# hyperparameters VI


What **Overfitting** means in a practical example

# Steps in Training a Deep Learning Model

1. Preparation of input data (training/test splitting, extraction of features and labels, conversion to numerical sequence structure)
2. Definition of the model (input, hidden and output layers)
3. Defining Evaluation Metrics
4. Training of the model
5. Prediction of validation data and evaluation of performance

# preparation

As before: tokenization and creation of the vocabulary.
Additionally **sequencing**:

```
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)
```

# model definition

An example of a neural network with pre-trained embeddings as input vector, two hidden layers (1 dense, 1 convolutional) and an output layer.

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

# Set evaluation metrics

In the next step, we define the loss-function we want to use (for binary classification tasks, usually *binary_crossentropy*), the learning rate, and the metrics that are displayed after each epoch

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

# train model

In the model initialization step, we add two more hyperparameters (*epochs* and *batch_Size*)

```
history <- model %>% fit(
x = x_train,
y = y_train,
epochs = 20,
batch_size = 128,
validation_data = list(x_test, y_test),
callbacks = list(stop_if_no_improvement)
)
```

# Prediction and evaluation

Finally, we predict and evaluate the data (e.g. using the *confusionMatrix* function of the *caret* package)

```
1  pred_results <- as.data.frame(predict(model,
       x_test))
2
3  # Classify observations with probability > 0.5 as 1
4  pred_results$pred <- ifelse(pred_results$V1>=0.5,
       1, 0)
5
6  # Combine predictions with the real annotated data
7  pred_results$true <- y_test
8
9  confusionMatrix(as.factor(pred_results$pred),
       as.factor(pred_results$true))
```

*... and now in* `R`

# More Resources

**Helpful Links**

- **Deep Learning Tutorial**: Practical Deep Learning
- what we haven't covered are Transformer models: **Tutorial on Transformer models in Colab** A practical introduction to fine-tuning these models by Moritz Laurer
- Overview of Hugging Face Language Models

# What we learned today

- ... How do we prepare our data for more complex machine learning models
- ... how do we build our own machine learning model

# Next session

- Last session of the seminar
  1. How to use LLM in R
  2. Clarification of open questions
  3. Reflection

# References I