Recap

Modulprüfung

- Anmeldung bis 06.07.
- Klausurfragerunde: 06.07.
- Klausur: 13.07.

WS:	Seminar	JH
SoSe:	Übung	NR
SoSe:	VL	NR

- Large Language Models
  - »Classical language models on steroids«
  - Learned Representation
  - Attention: Context tokens are not equally important
  - Two-phase training process
  - Scaling up data set sizes, processing power

Neural Networks

Sprachverarbeitung (VL + Ü)

Nils Reiter

July 4, 2023

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

From a Logistic Regression to a Neuron

- Hypothesis function of logistic regression:

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

Maps one value to another (just like many other functions)

From a Logistic Regression to a Neuron

- Hypothesis function of logistic regression:

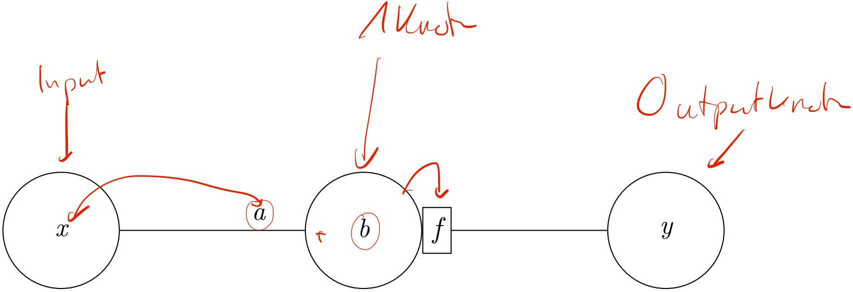
$$h(x) = \frac{1}{1 + e^{-(ax+b)}}$$

  Maps one value to another (just like many other functions)

- Further parameterization:

$$h(x) = \sigma(\underline{ax + b}) \qquad \text{with } \sigma(x) = \frac{1}{1 + e^{-x}}$$

What is a Neural Network?



$$y = \cancel{\sigma}\, f(\underline{ax + b})$$

Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

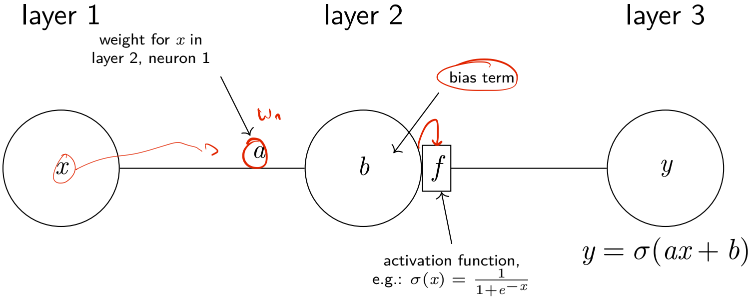


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

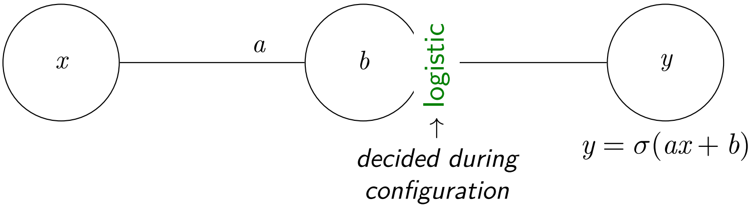


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

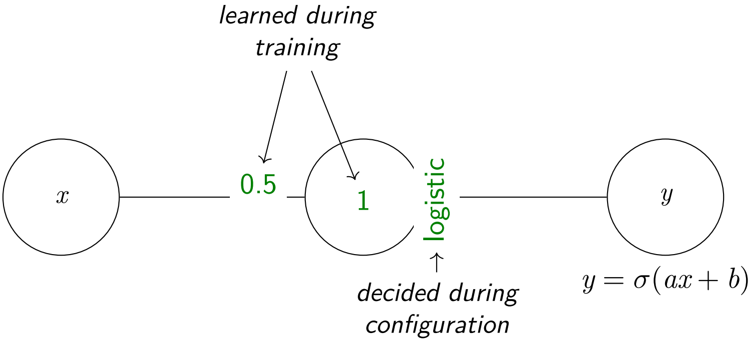


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

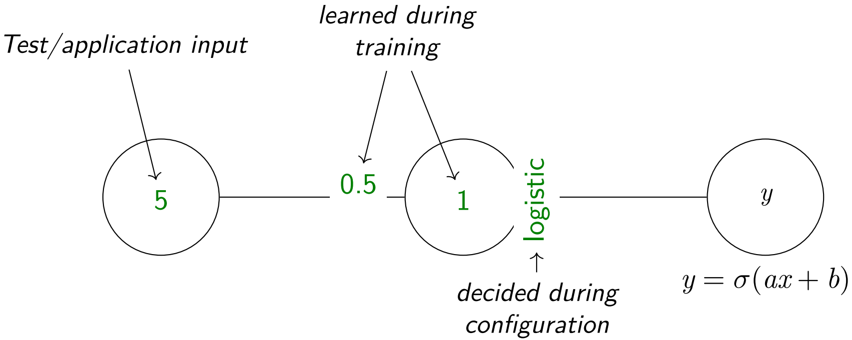


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

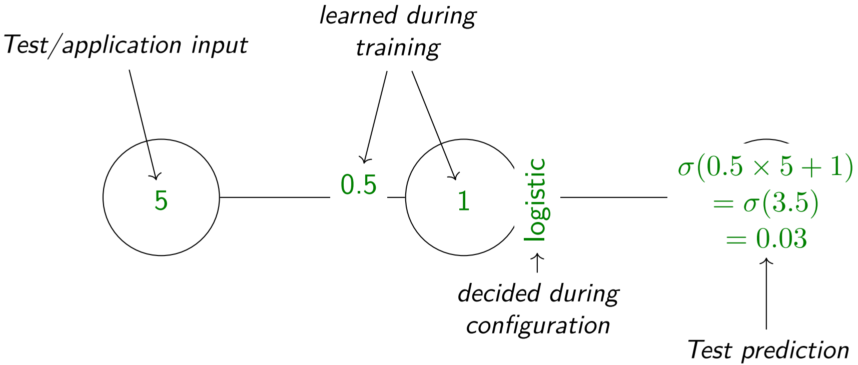


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Straightforward to extend to multiple features

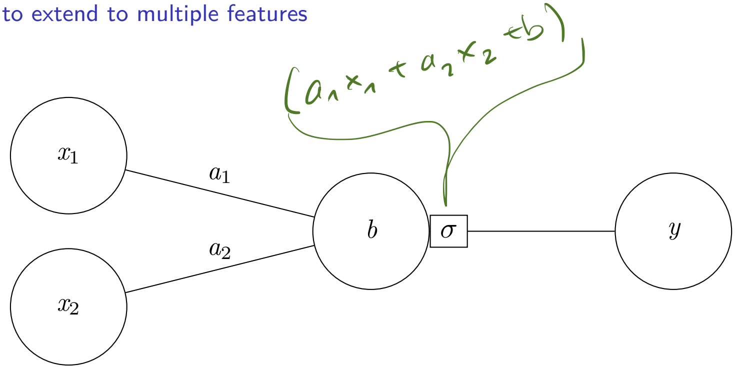
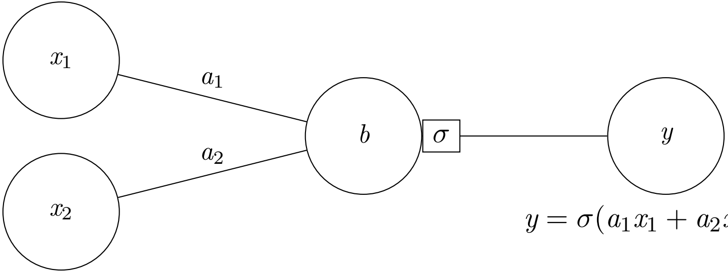


Figure: 1 neuron (with 2 features)

What is a Neural Network?

Straightforward to extend to multiple features



$$y = \sigma(a_1 x_1 + a_2 x_2 + b)$$

Figure: 1 neuron (with 2 features)

What is a Neural Network?

Straightforward to extend to multiple features and multiple regression nodes

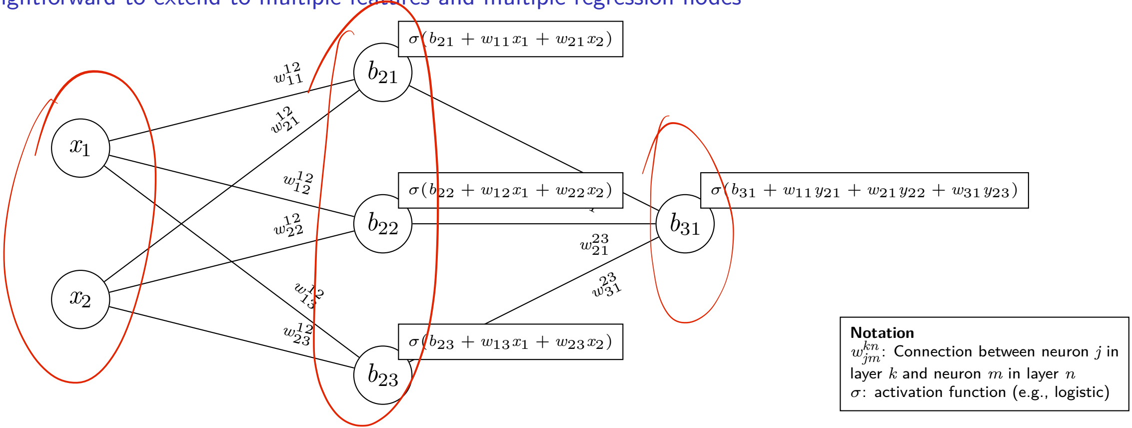


Figure: A simple neural network with 1 hidden layer

Prediction Model: Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)

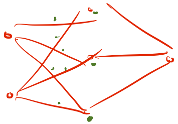
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Prediction Model: Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)
- Practically, a lot of the computation happens in matrices
  - Hidden layer
    - Weights from input to hidden: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$

Prediction Model: Forward Pass

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)
- Practically, a lot of the computation happens in matrices
  - Hidden layer
    - Weights from input to hidden: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = b_{21}, b_{22}, b_{23}$
- Hidden layer computation
  - $f_2(X) = \sigma((W_{1,2}^{\intercal} X) + B_2)$



Prediction Model: Forward Pass

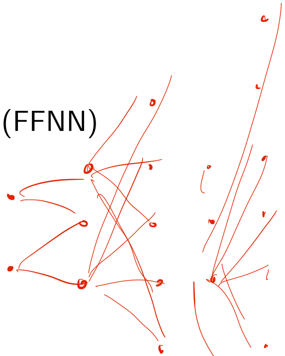
- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)
- Practically, a lot of the computation happens in matrices
  - Hidden layer
    - Weights from input to hidden: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$
- Hidden layer computation
  - $f_2(X) = \sigma((W_{1,2}^{\intercal} X) + B_2)$
- Deep learning involves a lot of matrix multiplication
  - GPUs are highly optimized for this
  - Hint: Gaming-GPUs that support CUDA are also usable for deep learning

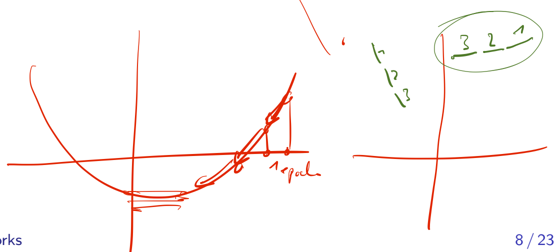
Feed-Forward Neural Networks

- The above is called a ›feed-forward neural network‹ (FFNN)
  - Information is fed only in forward direction

Feed-Forward Neural Networks

- The above is called a ›feed-forward neural network‹ (FFNN)
  - Information is fed only in forward direction
- Configuration choices
  - Activation function (next slide)
  - Layer size: Number of neurons in each layer
  - Number of layers
  - Loss function
  - Optimizer
- Training choices
  - Epochs/batches
  - Training status displays





Feed-Forward Neural Networks

Activation Functions

- All neurons of one layer have the same
- Popular choices:

logistic $y = \sigma(x) = \frac{1}{1+e^{-x}}$ – ›squashes‹ everything to a value between 0 and 1 sigmoid

- E.g.: $f([-0.5, 0.5, 1]) = [0.38, 0.62, 0.73]$

Feed-Forward Neural Networks

Activation Functions

- All neurons of one layer have the same
- Popular choices:

logistic $y = \sigma(x) = \frac{1}{1+e^{-x}}$ – ›squashes‹ everything to a value between 0 and 1

- E.g.: $f([-0.5, 0.5, 1]) = [0.38, 0.62, 0.73]$

relu $y = \max(0, x)$ – Makes everything negative to 0

- E.g.: $f([-0.5, 0.5, 1]) = [0, 0.5, 1]$

Feed-Forward Neural Networks

Activation Functions

- All neurons of one layer have the same
- Popular choices:

logistic $y = \sigma(x) = \frac{1}{1+e^{-x}}$ – ›squashes‹ everything to a value between 0 and 1

- E.g.: $f([-0.5, 0.5, 1]) = [0.38, 0.62, 0.73]$

relu $y = \max(0, x)$ – Makes everything negative to 0

- E.g.: $f([-0.5, 0.5, 1]) = [0, 0.5, 1]$

softmax Scales an entire vector such that elements sum to 1 (probability distribution)

- E.g.: $f([-0.5, 0.5, 1]) = [0.12, 0.33, 0.55]$

Training: »Backpropagation«

- Similar to gradient descent
- But
  - A lot more parameters
  - Weight updates need to be distributed over the layers
  - Because of multiple layers: Vanishing gradients
    - Backpropagation involves a lot of multiplication
    - Factors are between zero and one

    $\Rightarrow$ Numbers get very small very quickly

Training: »Backpropagation«

- Similar to gradient descent
- But
  - A lot more parameters
  - Weight updates need to be distributed over the layers
  - Because of multiple layers: Vanishing gradients
    - Backpropagation involves a lot of multiplication
    - Factors are between zero and one
    - $\Rightarrow$ Numbers get very small very quickly
- Training choice: Batches and epochs

Training a Feedforward Neural Network I

Stochastic Gradient Descent (SGD)

- Gradient Descent
  - Apply θ to all training instances
  - Calculate loss over entire data set
- Stochastic Gradient Descent
  - Data set in random order
  - Calculate loss for every single instance, then update weights

Training a Feedforward Neural Network II

When to stop the training

- Logistic regression: Stop in minimum
- In theory, that's what we want
- In practice
  - We usually are not exactly in the minimum
  - It's not important to be exactly in the minimum

$\Rightarrow$ Fixed number of iterations over the data set (= number of epochs)

Batches vs. Epochs

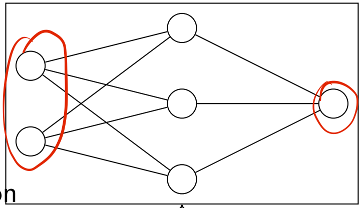
batch Number of instances used before updating weights

epochs Number of iterations over all instances

Dimensions

- Dimensionality of neural networks major source of confusion

Dimensions



- Dimensionality of neural networks major source of confusion
- In this example
  - Single input object represented with two numbers (= 1D)
  - Output is a single number
- Dimensions:
  - Input data set: 2D (because multiple instances)
  - Output data: 1D (a list of single numbers)

$[\; x_1 \quad x_2 \;] \Rightarrow 1D$

$$\left[\begin{matrix} [\, x & x \,] \\ [\, x & x \,] \\ [\, x & x \,] \end{matrix} \right]$$

Section 1

Practical Deep Learning

Libraries

- Most developments take place in Python
- Deep learning in python rests on several independent libraries
  - `numpy` Provides efficient matrices and arrays
  - `pandas` Convenient working with tabular data (inspired by `data.frames` in R)
  - `scikit-learn` ›Classical‹ machine learning (not deep learning)
  - `tensorflow` Basic, low-level machine learning and math
  - `keras` High-level deep learning (built on top of tensorflow)
  - `pytorch` Newer alternative to tensorflow
  - `transformers` Library for transformer models by Hugging Face

Libraries

- Most developments take place in Python
- Deep learning in python rests on several independent libraries
  - `numpy` Provides efficient matrices and arrays
  - `pandas` Convenient working with tabular data (inspired by `data.frames` in R)
  - `scikit-learn` ›Classical‹ machine learning (not deep learning)
  - `tensorflow` Basic, low-level machine learning and math
  - `keras` High-level deep learning (built on top of tensorflow)
  - `pytorch` Newer alternative to tensorflow
  - `transformers` Library for transformer models by Hugging Face
- Documentation is fragmented – important links:
  - `https://keras.io/api/`
  - `https://pandas.pydata.org/docs/reference/index.html`
  - `https://scikit-learn.org/stable/modules/classes.html`
  - `https://huggingface.co/docs/transformers/index`

keras

- High-level Python API for deep learning
    1. Adaptations exist for R, Java, JavaScript, ...
- Built on top of tensorflow

keras

- High-level Python API for deep learning
  1. Adaptations exist for R, Java, JavaScript, ...
- Built on top of tensorflow
- Pattern
  1. Loading and preprocessing data ← das dauert am längsten
  2. Layout the network
  3. Set hyper parameters
  4. Run training

Configuration

- Sequential API: Linear topology of layers
- Functional API: Graph of layers

Configuration

- Sequential API: Linear topology of layers
- Functional API: Graph of layers

Listing 3: Sequential API

```
# model layout
model = Sequential()
model.add(...)
model.add(...)

# hyperparameter specification
model.compile(loss=...,
  optimizer=...)

# training
model.fit(..., epochs=...,
  batch_size=...)
```

Listing 4: Functional API

```
# model layout
in = ...
out = Dense(10)(in)
model = Model(inputs=in,
  outputs=out)

# hyperparameter specification
model.compile(loss=...,
  optimizer=...)

# training
model.fit(..., epochs=...,
  batch_size=...)
```

Configuration

Two most basic layer types

- Dense: »Just your regular densely-connected NN layer.«
  - https://keras.io/api/layers/core_layers/dense/

```
layer = Dense(3, # number of neurons
  activation = activations.sigmoid, # activation function
  name = "dense layer 7" # useful for debugging/visualisation
  ... # more options, see docs
)
```

- Input: Marks layers to accept data
  - https://keras.io/api/layers/core_layers/input/

```
layer = Input(shape=(15,) # number of input dimensions/features
  name = "input layer", # useful for debugging/visualisation
  ... # see docs
)
```

Shape

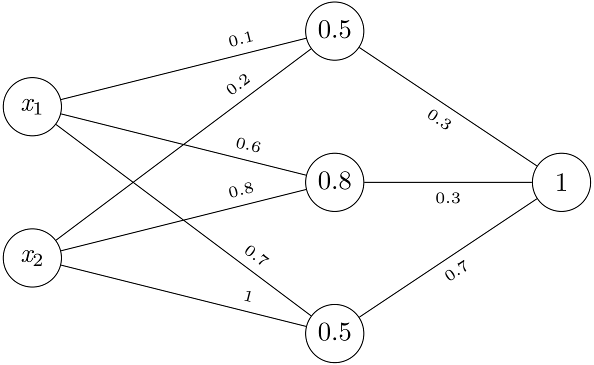
- Description of the dimensionality of the data
- A vector of numbers, giving the number of elements for each dimension
- Python tuple
  - List with fixed length: `x = (5,3,1) #a tuple`
  - ⚠ Tuple with one element printed as `(5,)` or `5`

Shape

- Description of the dimensionality of the data
- A vector of numbers, giving the number of elements for each dimension
- Python tuple
  - List with fixed length: `x = (5,3,1) #a tuple`
  - ⚠ Tuple with one element printed as `(5,)` or `5`

```
x = np.zeros(5) # array([0., 0., 0., 0., 0.])
x.shape # returns (5,)
x = np.zeros((3,5))
# array([[0., 0., 0., 0., 0.],
#        [0., 0., 0., 0., 0.],
#        [0., 0., 0., 0., 0.]])
x.shape # returns (3,5)
```

Example



x_1	x_2	y
0	0	0.86169636
1	0	0.87786007
1	1	0.891605
10	10	0.90814614
$\vdots$	$\vdots$	$\vdots$

Figure: Neural network with randomly initialized weights

```python
# setup the model architecture
model = Sequential()
model.add(InputLayer(input_shape=(2,)))
model.add(Dense(3, activation="sigmoid"))
model.add(Dense(1, activation="sigmoid"))

model.compile() # compile it

w1 = [ # weights between neurons
  np.array([[0.1,0.6,0.7],[0.2,0.8,1]]),
  # bias terms
  np.array([0.5,0.8,0.5]) ]

w2 = [ # weights between neurons
  np.array([[0.3],[0.3],[0.7]]),
  # bias terms
  np.array([1]) ]

model.layers[0].set_weights(w1)
model.layers[1].set_weights(w2)

y = model.predict(np.array([[0,0]])) # generate predictions
print(y)
```

Neural network with manually specified weights as above on lehre.idh: `simple-nn.py`

Section 2

Exercise

Exercise

- Task 1