Recap: Machine Learning

Naive Bayes

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

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

- Training: Count relative frequencies

Logistic Regression

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

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

- Training: Gradient descent with loss function

Machine Learning 3: Neural Networks

VL Sprachliche Informationsverarbeitung

Nils Reiter
nils.reiter@uni-koeln.de

December 8, 2022
Winter term 2022/23

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:

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

What is a Neural Network?

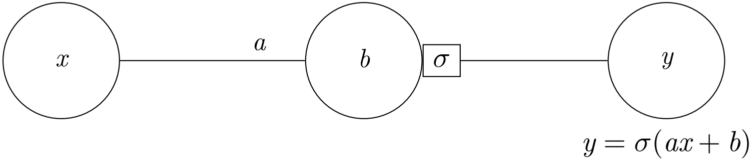


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

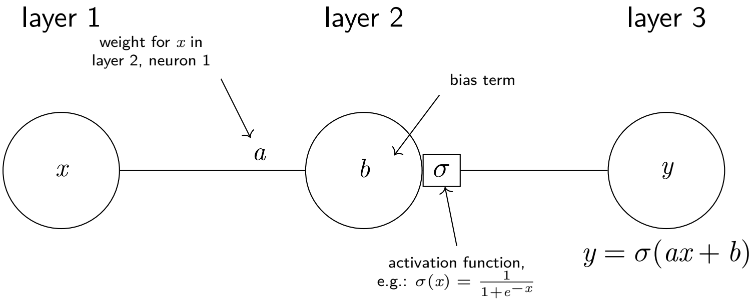


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

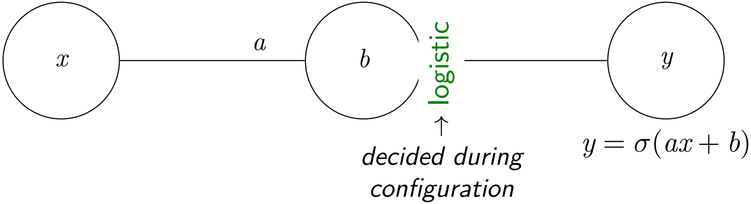


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

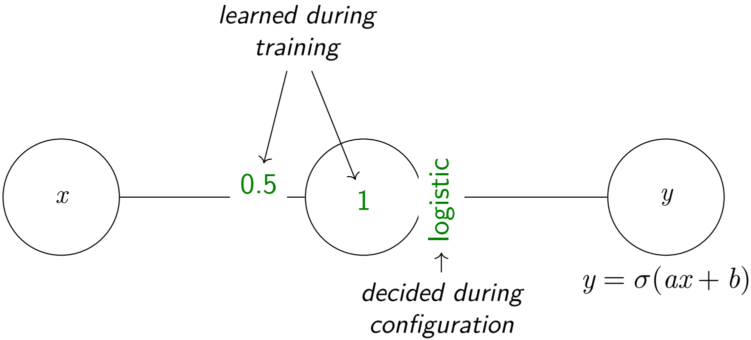


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

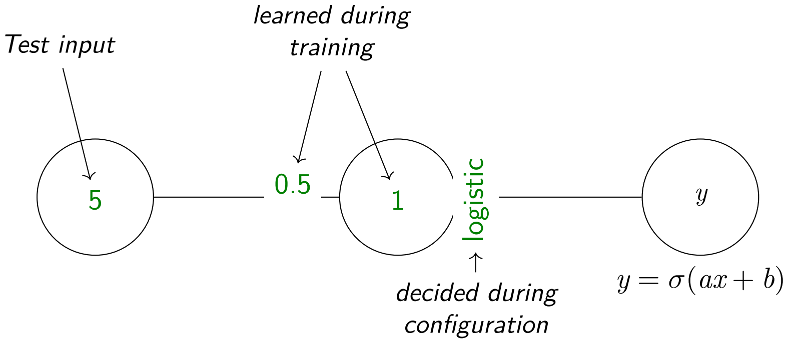


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

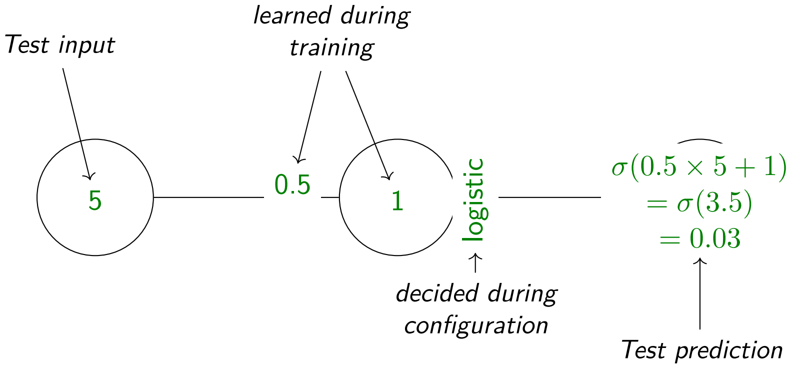


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Straightforward to extend to multiple features

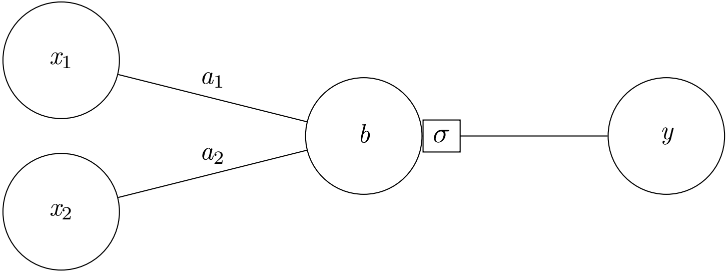


Figure: 1 neuron (with 2 features)

What is a Neural Network?

Straightforward to extend to multiple features

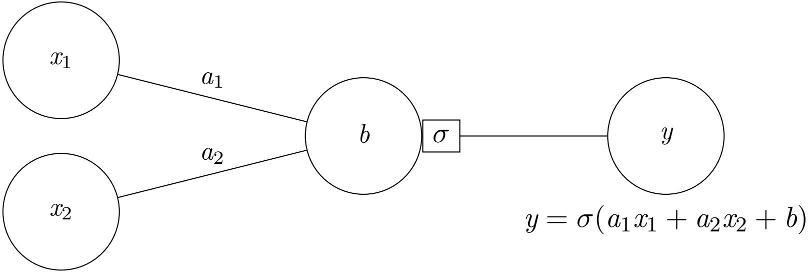
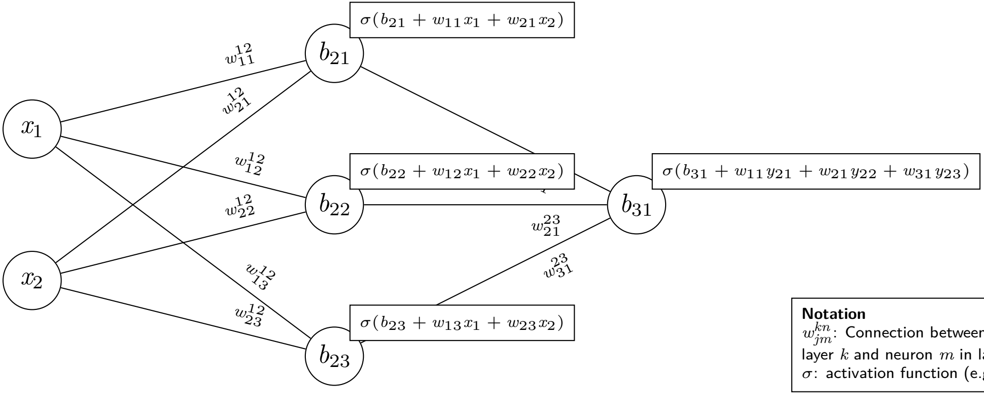


Figure: 1 neuron (with 2 features)

What is a Neural Network?

Straightforward to extend to multiple features and multiple regression nodes



Notation
w_{jm}^{kn}: Connection between neuron j in layer k and neuron m in layer n
σ: activation function (e.g., logistic)

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)

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
- **relu** $y = \max(0, x)$ – Makes everything negative to 0
- **softmax** Scales a vector such that values sum to 1 (probability distribution)

Training: "Backpropagation"

- Similar to gradient descent
- But
  - A lot more parameters
  - 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
  - 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 (last week): 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

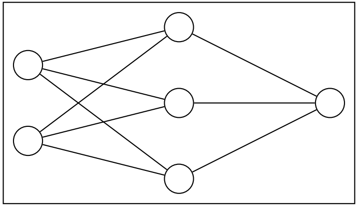
$\Rightarrow$ Fixed number of iterations over the data set (= number of epochs)

Batches vs. Epochs

batch Number of instances used before updating weights

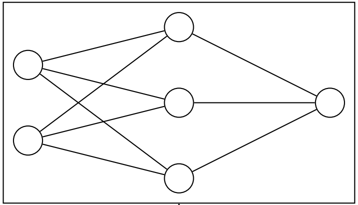
epochs Number of iterations over all instances

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
- Entire input data set: 2D (because multiple instances)

Section 2

Practical Deep Learning

Libraries

- 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
- Libraries are well integrated

Libraries

- 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
- Libraries are well integrated
- 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`

keras

- https://keras.io
- High-level Python API for deep learning
- Built on top of tensorflow
- Pattern
    1. Layout the network
    2. Set hyper parameters
    3. Run training

Listing 1: Installing Keras

```
pip install keras
```

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 4: Sequential API

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

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

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

Listing 5: 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)
```

A Full Example

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

# create a random data set
train = np.random.randn(100)
train = train.reshape([4,25])
y_train = train[0]
x_train = np.rot90(train[1:])

# setup the model architecture
model = keras.Sequential()
model.add(layers.Input(shape=(3,)))
model.add(layers.Dense(5, activation="sigmoid"))
model.add(layers.Dense(1, activation="softmax"))

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

# train it
model.fit(x_train, y_train, epochs=100, batch_size=5)
```

Feedforward Neural Networks

Code

```
# network architecture
model = Sequential()
model.add(layers.Dense(5, input_shape=(3,), activation="sigmoid"))
model.add(layers.Dense(1, activation="sigmoid"))

# training configuration
model.compile(loss="binary_crossentropy",
  optimizer="sgd", # = stochastic gradient descent
  metrics=["accuracy"])

# training
model.fit(train_x, train_y, epochs=150, batch_size=2,
  verbose=1)
```