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

UNIVERSITÄT
ZU KÖLN

Machine Learning: Gradient Descent & Neural Networks

VL Sprachliche Informationsverarbeitung

Nils Reiter
nils.reiter@uni-koeln.de

December 12, 2024
Winter term 2024/25

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 3

Gradient Descent

Learning Regression Models

- How to select the parameters w_0, w_1 such that the hypothesis function describes the data points as best as possible?
- Learning algorithm *Gradient Descent*

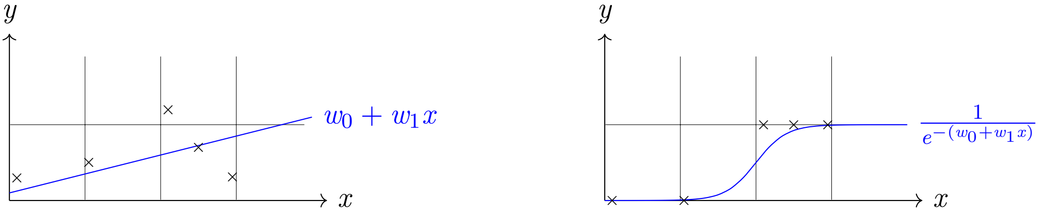
Learning Regression Models

- How to select the parameters w_0, w_1 such that the hypothesis function describes the data points as best as possible?
- Learning algorithm *Gradient Descent*

SPOILER ALERT! Gradient descent is half of the algorithms of deep learning

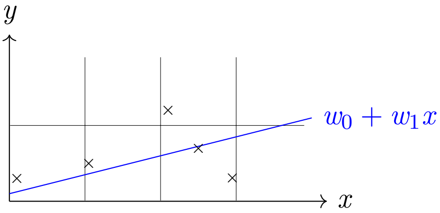
Loss: Intuition

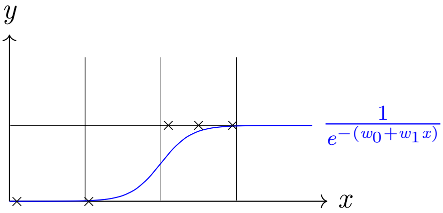
The *loss* measures the 'wrongness' of values for w_0 and w_1



Loss: Intuition

The *loss* measures the ‘wrongness’ of values for w_0 and w_1

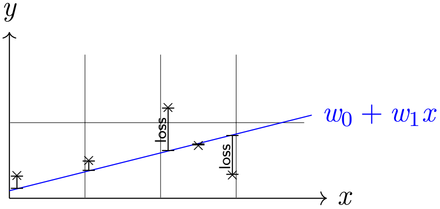


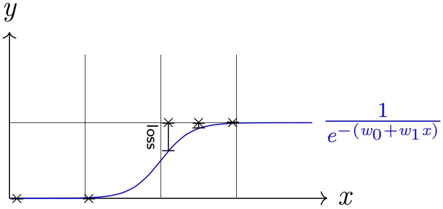


- How big is the gap between a hypothesis (= the) and the data?
- Is $\vec{w} = \langle 0.3, 0.5 \rangle$ or $\vec{w} = \langle 0.4, 0.4 \rangle$ better?

Loss: Intuition

The *loss* measures the 'wrongness' of values for w_0 and w_1



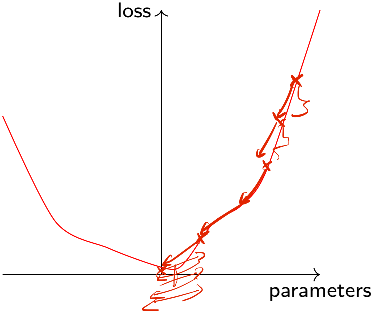


- How big is the gap between a hypothesis (= the) and the data?
- Is $\vec{w} = \langle 0.3, 0.5\rangle$ or $\vec{w} = \langle 0.4, 0.4\rangle$ better?

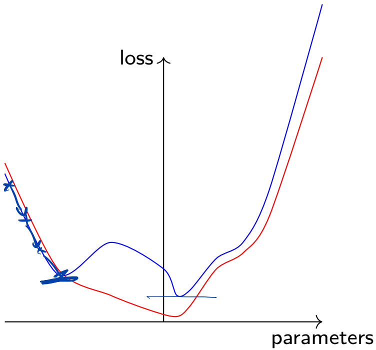
Gradient Descent: Intuition

- Loss should be as small as possible
- Total loss can be calculated for given parameters $\vec{w} = (w_0, w_1)$ (and with respect to a data set!)
- Idea:
  - Make many iterations
  - In each iteration, change $\vec{w}$ towards to make the loss smaller
  - We use the derivative of the loss function to know how $\vec{w}$ needs to be changed

Loss function: Intuition



Loss function: Intuition



Function should be convex!
If not, we might get stuck in local minimum

Loss Function with More Parameters

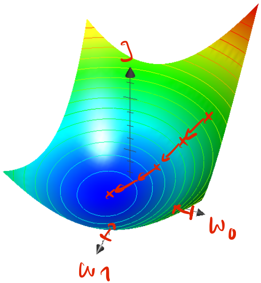


Figure: The loss function in our setting visualised

- Searching for the a, b settings with minimal loss
- $=$ Searching for the minimum!

Hypothesis vs. Loss Function

Hypothesis function h

- Calculates outcome for a single instance based on
  - Feature values $\vec{x}$
  - Parameter values $\vec{w}$

Loss function J

- Calculates ‘wrongness’ of h based on
  - Parameter values $\vec{w}$
  - A data set consisting of many instances with
    - Feature values $\vec{x}$
    - Correct outcomes Y

Loss Function

Definition

Loss function depends on hypothesis function

Linear hypothesis function

- $h(x) = w_0 + w_1 x_1$
- Loss: Mean squared error

Loss Function

Definition

Loss function depends on hypothesis function

Linear hypothesis function

- $h(x) = w_0 + w_1 x_1$
- Loss: Mean squared error

Logistic hypothesis function

- $h(x) = \sigma(w_0 + w_1 x_1) = \frac{1}{e^{-(w_0 + w_1 x_1)}}$
- Loss: (Binary) cross-entropy loss

Loss Function

Definition for Linear Regression

- The loss function is a function on parameter values a and b (for a given hypothesis function and data set)
  - Hypothesis function: $h_{\vec{w}} = w_1 x + w_0$

$\vec{w} = (w_0, w_1)$: parameters $h_{\vec{w}}$: hypothesis function m: number of items

$$J(\vec{w}) =$$

Loss Function

Definition for Linear Regression

- The loss function is a function on parameter values a and b
  (for a given hypothesis function and data set)
  - Hypothesis function: $h_{\vec{w}} = w_1 x + w_0$

$\vec{w} = (w_0, w_1)$: parameters $h_{\vec{w}}$: hypothesis function m: number of items

$$J(\vec{w}) = \underline{h_{\boxed{\vec{w}}}(x_{\boxed{i}})} - \boxed{y_{\boxed{i}}}$$

- Calculate the loss for item i

Loss Function

Definition for Linear Regression

- The loss function is a function on parameter values a and b
  (for a given hypothesis function and data set)
  - Hypothesis function: $h_{\vec{w}} = w_1 x + w_0$

$\vec{w} = (w_0, w_1)$: parameters $h_{\vec{w}}$: hypothesis function m: number of items

$$J(\vec{w}) = \qquad \underline{(h_{\vec{w}}(x_i) - y_i})^2$$

- Calculate the loss for item i
- Square the error

Loss Function

Definition for Linear Regression

- The loss function is a function on parameter values a and b
  (for a given hypothesis function and data set)
  - Hypothesis function: $h_{\vec{w}} = w_1 x + w_0$

$\vec{w} = (w_0, w_1)$: parameters $h_{\vec{w}}$: hypothesis function m: number of items

$$J(\vec{w}) = \sum_{i=1}^{m} (h_{\vec{w}}(x_i) - y_i)^2$$

- Calculate the loss for item i
- Square the error
- Sum them up

Loss Function

Definition for Linear Regression

- The loss function is a function on parameter values a and b
  (for a given hypothesis function and data set)
  - Hypothesis function: $h_{\vec{w}} = w_1 x + w_0$

$\vec{w} = (w_0, w_1)$: parameters $h_{\vec{w}}$: hypothesis function m: number of items

$$J(\vec{w}) = \frac{1}{m} \sum_{i=1}^{m} (h_{\vec{w}}(x_i) - y_i)^2$$

- Calculate the loss for item i
- Square the error
- Sum them up
- Divide by the number of items
  - Known as: *Mean squared error*

Loss Function

Definition for Linear Regression

- The loss function is a function on parameter values a and b (for a given hypothesis function and data set)
  - Hypothesis function: $h_{\vec{w}} = w_1 x + w_0$

$\vec{w} = (w_0, w_1)$: parameters $h_{\vec{w}}$: hypothesis function m: number of items

$$J(\vec{w}) = \frac{1}{2}\frac{1}{m}\sum_{i=1}^{m}(h_{\vec{w}}(x_i) - y_i)^{2}$$

2 . . .

- Calculate the loss for item i
- Square the error
- Sum them up
- Divide by the number of items
  - Known as: *Mean squared error*
- Divide by two
  - out of convenience, because derivation

Loss function

Definition for Logistic Regression

- Two cases: $y_i = 0$ or $y_i = 1$ – y_i: real outcome for instance i
  - I.e., for each instance, one of the summands is 0

Loss function

Definition for Logistic Regression

- Two cases: $y_i = 0$ or $y_i = 1$ – y_i: real outcome for instance i
  - I.e., for each instance, one of the summands is 0

$$J(\vec{w}) = \qquad h_{\vec{w}}(x_i) + \qquad (1 - h_{\vec{w}}(x_i))$$

Loss function

Definition for Logistic Regression

- Two cases: $y_i = 0$ or $y_i = 1$ – y_i: real outcome for instance i
  - I.e., for each instance, one of the summands is 0

$$J(\vec{w}) = \qquad\qquad \log h_{\vec{w}}(x_i) + \qquad\qquad \log\left(1 - h_{\vec{w}}(x_i)\right)$$

Loss function

Definition for Logistic Regression

- Two cases: $y_i = 0$ or $y_i = 1$ – y_i: real outcome for instance i
  - I.e., for each instance, one of the summands is 0

$$J(\vec{w}) = \quad y_i \log h_{\vec{w}}(x_i) + (1 - y_i) \log (1 - h_{\vec{w}}(x_i))$$

$$\begin{cases} \log h(x_i) & \text{if } y_i = 1 \\ \log(1 - h(x_i)) & \text{if } y_i = 0 \end{cases}$$

Loss function

Definition for Logistic Regression

- Two cases: $y_i = 0$ or $y_i = 1$ – y_i: real outcome for instance i
  - I.e., for each instance, one of the summands is 0

$$J(\vec{w}) = \enclose{circle}{-} \enclose{circle}{\frac{1}{m} \sum_{i=0}^{m}} \underline{y_i} \log h_{\vec{w}}(x_i) + (1 - y_i) \log (1 - h_{\vec{w}}(x_i))$$

Loss function

Definition for Logistic Regression

- Two cases: $y_i = 0$ or $y_i = 1$ – y_i: real outcome for instance i
  - I.e., for each instance, one of the summands is 0

$$J(\vec{w}) = -\frac{1}{m}\sum_{i=0}^{m} \underbrace{y_i \log h_{\vec{w}}(x_i)}_{0 \text{ iff } y_i=0} + \underbrace{(1-y_i)\log(1-h_{\vec{w}}(x_i))}_{0 \text{ iff } y_i=1}$$

Loss function

Definition for Logistic Regression

- Two cases: $y_i = 0$ or $y_i = 1$ – y_i: real outcome for instance i
    - I.e., for each instance, one of the summands is 0

y_i	$h_{\vec{w}}(x_i) + \epsilon$	$y_i \log h_{\vec{w}}(x_i) + (1 - y_i) \log(1 - h_{\vec{w}}(x_i))$
0	1.0000001	-23.2535
0	0	0

Loss function

Definition for Logistic Regression

- Two cases: $y_i = 0$ or $y_i = 1$ – y_i: real outcome for instance i
  - I.e., for each instance, one of the summands is 0

y_i	$h_{\vec{w}}(x_i) + \epsilon$	$y_i \log h_{\vec{w}}(x_i) + (1 - y_i) \log(1 - h_{\vec{w}}(x_i))$
0	1.0000001	-23.2535
0	0	0
1	1	0
1	0.0000001	-23.2535

Loss function

Definition for Logistic Regression

- Two cases: $y_i = 0$ or $y_i = 1$ ~~– y_i~~: real outcome for instance i → y_i
  - I.e., for each instance, one of the summands is 0

y_i	$h_{\vec{w}}(x_i) + \epsilon$	$y_i \log h_{\vec{w}}(x_i) + (1 - y_i) \log(1 - h_{\vec{w}}(x_i))$
0	1.0000001	−23.2535
0	0	0
1	1	0
1	0.0000001	−23.2535
1	0.8 (0,2)	−0.3219281
1	0.2	−2.321928

Caveat: $\log 0$ is undefined – add $\epsilon = 0.0000001$ if needed

Misunderstandings about Loss

- Loss values are not an evaluation measure
- It is meaningless to report loss values in the same way as accuracy or precision

Misunderstandings about Loss

- Loss values are not an evaluation measure
- It is meaningless to report loss values in the same way as accuracy or precision
- Because it's not scaled
- When used to compare, scale not important

Misunderstandings about Loss

- Loss values are not an evaluation measure
- Loss has no meaning outside of the current experiment
- Because it's not scaled
- When used to compare, scale not important

Example

x	y
10	3
105	5
150	8
250	7
295	13

Table: Lower Loss Value

x	y
10	30
105	50
150	80
250	70
295	130

Table: Higher Loss Value

Section 2

Neural Networks

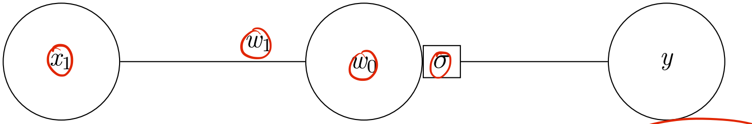
From a Logistic Regression to a Neuron

- Hypothesis function of logistic regression:

$$h(x) = \sigma(w_0 + w_1 x_1) \qquad \text{with } \sigma(x) = \frac{1}{1 + e^{-x}}$$

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

What is a Neural Network?



$$y = \sigma(w_0 + w_1 x)$$

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

What is a Neural Network?

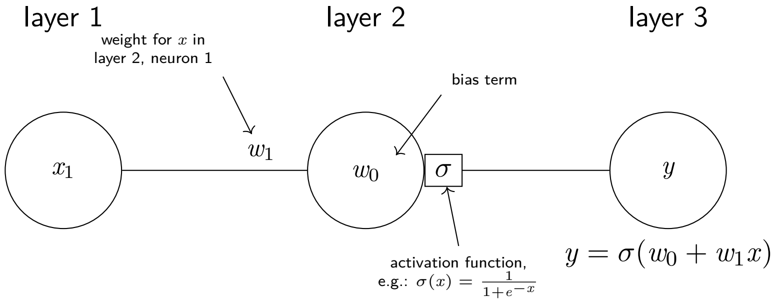

layer 1
layer 2
layer 3
weight for x in layer 2, neuron 1
bias term
x_1
w_1
w_0
σ
y
activation function, e.g.: $\sigma(x) = \frac{1}{1+e^{-x}}$
$y = \sigma(w_0 + w_1 x)$

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

What is a Neural Network?

Example

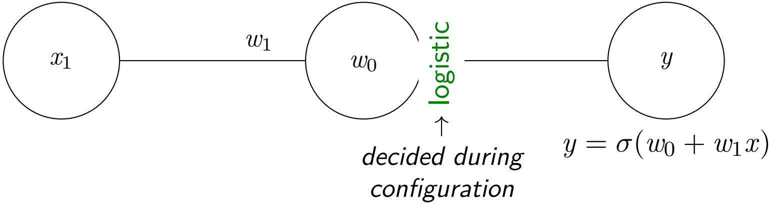


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

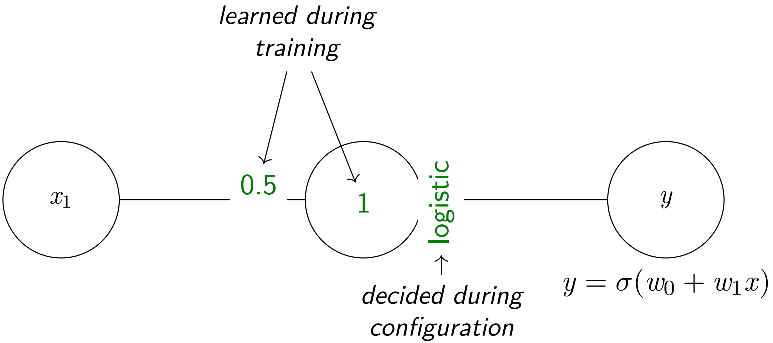

learned during training
x_1
0.5
1
logistic
y
decided during configuration
$y = \sigma(w_0 + w_1 x)$

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

What is a Neural Network?

Example

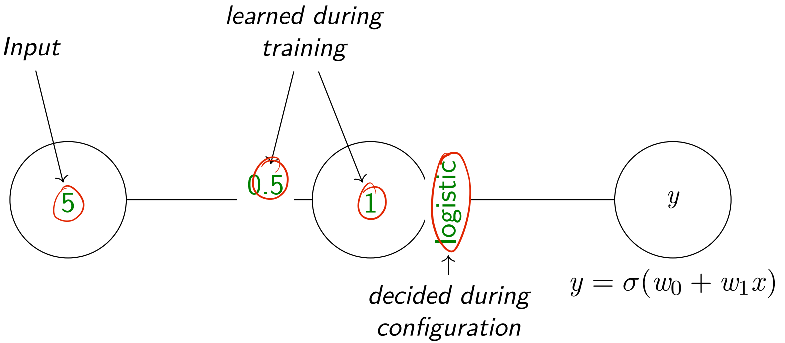


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Example

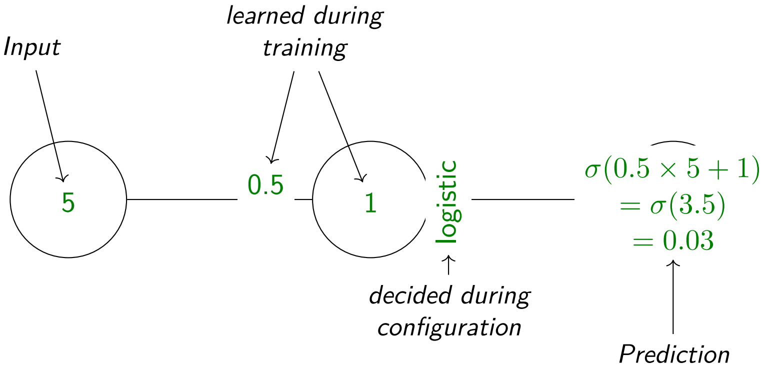


Figure: 1 neuron (with logistic activation) = logistic regression (with 1 feature)

What is a Neural Network?

Straightforward to extend to multiple features

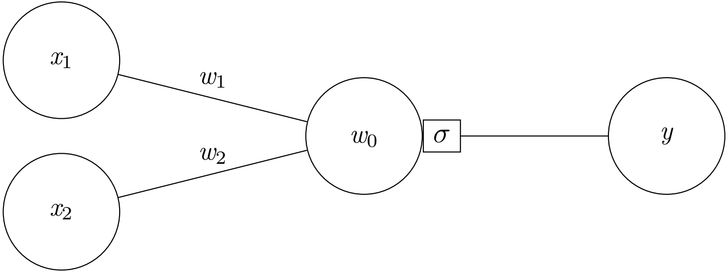
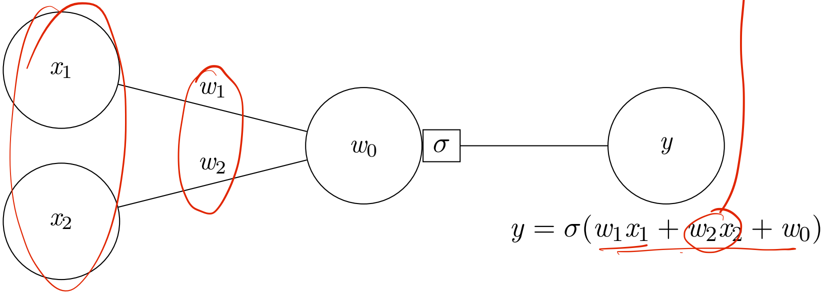


Figure: 1 neuron (with 2 features)

What is a Neural Network?

Straightforward to extend to multiple features

$$y = \sigma(w_1 x_1 + w_0)$$

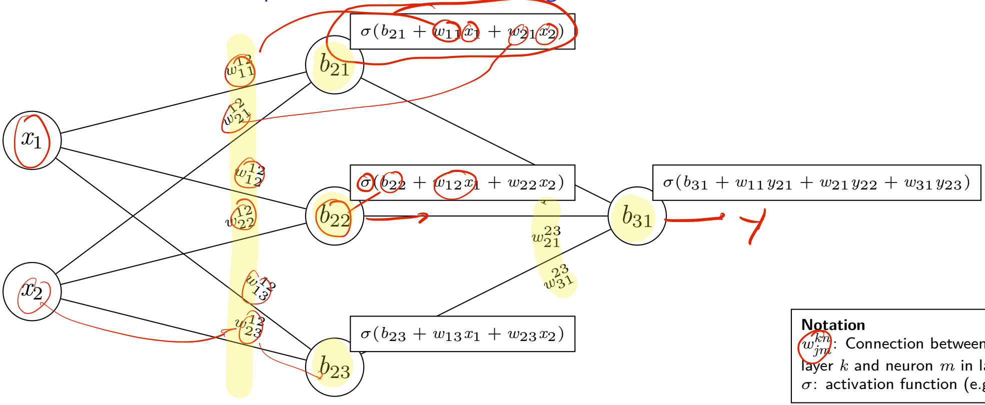


$$y = \sigma(w_1 x_1 + w_2 x_2 + w_0)$$

Figure: 1 neuron (with 2 features)

What is a Neural Network?

Straightforward to extend to multiple features and multiple regression nodes



$\sigma(b_{21} + w_{11}x_1 + w_{21}x_2)$

$\sigma(b_{22} + w_{12}x_1 + w_{22}x_2)$

$\sigma(b_{23} + w_{13}x_1 + w_{23}x_2)$

$\sigma(b_{31} + w_{11}y_{21} + w_{21}y_{22} + w_{31}y_{23})$

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 (and 13 parameters)

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

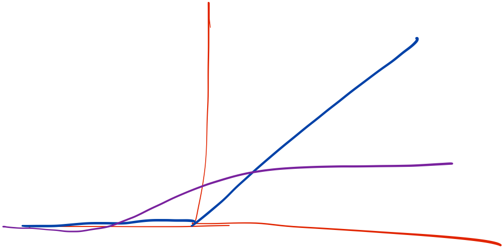
- The above is called a 'feed-forward neural network' (FFNN)
  - Information is fed only in forward direction
- Configuration choices
  - Activation function (next slide)
  - Layer size: Number of neurons in each layer
  - Number of layers
  - Loss function
  - Optimizer
- Training choices
  - Epochs/batches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

demo

playground.tensorflow.org

Feed-Forward Neural Networks

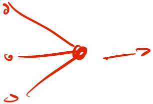
Activation Functions



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)



$y \in [0;1]$

- Spam 0-1
- Nicht-Spam 0-1
- Donut 0-1

$\Rightarrow \Sigma = 1$

Training: “Back Propagation”

- Similar to gradient descent
- But
  - A lot more parameters
  - Because of multiple layers: Vanishing gradients
    - Back propagation involves a lot of multiplication
    - Factors are between zero and one

    ⇒ Numbers get very small very quickly

Training: “Back Propagation”

- Similar to gradient descent
- But
  - A lot more parameters
  - Because of multiple layers: Vanishing gradients
    - Back propagation involves a lot of multiplication
    - Factors are between zero and one

    ⇒ 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

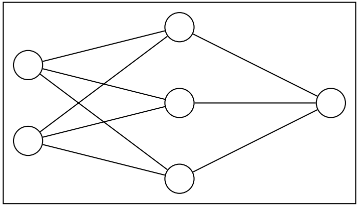
⇒ Fixed number of iterations over the data set (= number of epochs)

Batches vs. Epochs

batch Number of instances used before updating weights

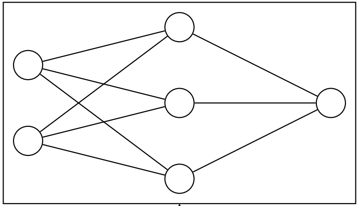
epochs Number of iterations over all instances

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 3

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)
```

demo

Python

Section 4

Summary

Summary

Gradient Descent

- Initialise $\vec{w}$ with random values (e.g., 0)
- Repeat:
  - Find the direction to the minimum by taking the derivative
  - Change $\vec{w}$ accordingly, using a learning rate η
  - Stop when $\vec{w}$ don't change anymore

Neural Networks

- Single neuron: 1 logistic regression
- Network: Many neurons
- Practical aspects in Python