Fundamentals of Neural Networks

Mathias Jackermeier

June 12, 2018

Technische Universität München

2018-06-11

Fundamentals of Neural Networks

Fundamentals of Neural Networks

Mathias Jackermeier
June 12, 2018
Technische Universität München

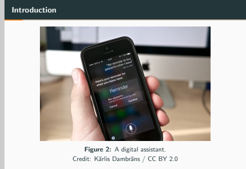
Zeitung ⇒ Künstliche Intelligenz, Gedichte schreiben SZ ⇒ Deep Learning

Introduction

Figure 1: A self-driving car.
Credit: Marc van der Chijs / CC BY-ND 2.0

Introduction

Figure 2: A digital assistant.
Credit: Kārlis Dambrāns / CC BY 2.0



Jedes intelligente System benutzt neuronale Netze

The Perceptron

The Perceptron

MNIST Data Sample

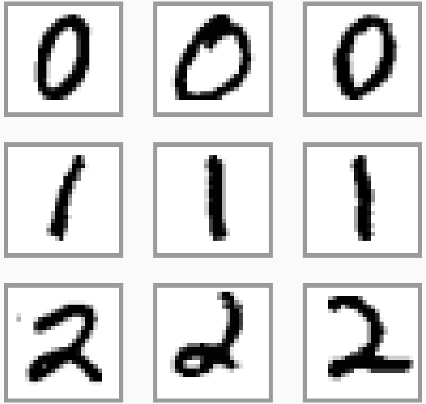
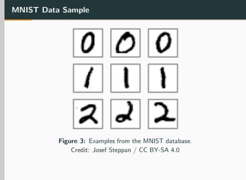


Figure 3: Examples from the MNIST database.
Credit: Josef Steppan / CC BY-SA 4.0



Example Task

- Predict whether an input image of a handwritten digit shows a zero or another digit

Example Task
- Predict whether an input image of a handwritten digit shows a zero or another digit

Example Task

- Predict whether an input image of a handwritten digit shows a zero or another digit
- The image is represented as a flattened vector of pixel intensities $\mathbf{x} \in \mathbb{R}^{784}$

Example Task

- Predict whether an input image of a handwritten digit shows a zero or another digit
- The image is represented as a flattened vector of pixel intensities $\mathbf{x} \in \mathbb{R}^{784}$

Example Task

- Predict whether an input image of a handwritten digit shows a zero or another digit
- The image is represented as a flattened vector of pixel intensities $\mathbf{x} \in \mathbb{R}^{784}$
- The output should be 1 if the image shows a zero, otherwise it should be -1

Example Task

- Predict whether an input image of a handwritten digit shows a zero or another digit
- The image is represented as a flattened vector of pixel intensities $\mathbf{x} \in \mathbb{R}^{784}$
- The output should be 1 if the image shows a zero, otherwise it should be -1

Example Task

- Predict whether an input image of a handwritten digit shows a zero or another digit
- The image is represented as a flattened vector of pixel intensities $\mathbf{x} \in \mathbb{R}^{784}$
- The output should be 1 if the image shows a zero, otherwise it should be -1
- **Idea**: Assign a weight to every input pixel

Example Task

- Predict whether an input image of a handwritten digit shows a zero or another digit
- The image is represented as a flattened vector of pixel intensities $\mathbf{x} \in \mathbb{R}^{784}$
- The output should be 1 if the image shows a zero, otherwise it should be -1
- **Idea**: Assign a weight to every input pixel

Model Specification

The perceptron accepts n input values and computes an output value $\hat{y}$:

$$
\begin{aligned}
\hat{y} &= \operatorname{sign}\left(\sum_{i=1}^{n} w_i x_i\right) \\
\equiv \hat{y} &= \operatorname{sign}\left(\mathbf{w}^\top \mathbf{x}\right)
\end{aligned}
\tag{1}
$$

Model Specification

The perceptron accepts n input values and computes an output value $\hat{y}$:

$$
\begin{aligned}
\hat{y} &= \operatorname{sign}\left(\sum_{i=1}^{n} w_i x_i\right) \\
\equiv \hat{y} &= \operatorname{sign}\left(\mathbf{w}^\top \mathbf{x}\right)
\end{aligned}
\tag{1}
$$

Visual Representation

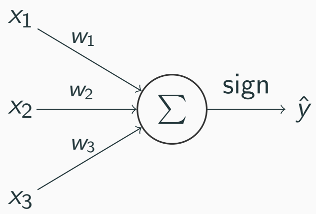
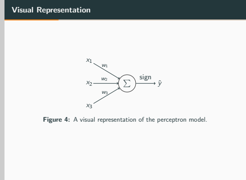


Figure 4: A visual representation of the perceptron model.



1950

Neuron im Gehirn: Nervenzelle Eingaben Ausgaben

Generalizations

- The perceptron is often used in a modified form

Generalizations
- The perceptron is often used in a modified form

f: Aktivierungsfunktion

Generalizations

- The perceptron is often used in a modified form
- A scalar bias value can be added to the output computation:

$$\hat{y} = \text{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{2}$$

Generalizations
- The perceptron is often used in a modified form
- A scalar bias value can be added to the output computation:
$$\hat{y} = \text{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{2}$$

f: Aktivierungsfunktion

Generalizations

- The perceptron is often used in a modified form
- A scalar bias value can be added to the output computation:

$$\hat{y} = \text{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{2}$$

- The sign function can be replaced with a generic function f:

$$\hat{y} = f\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{3}$$

Generalizations
- The perceptron is often used in a modified form
- A scalar bias value can be added to the output computation:
$$\hat{y} = \text{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{2}$$
- The sign function can be replaced with a generic function f:
$$\hat{y} = f\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{3}$$

f: Aktivierungsfunktion

Generalizations

- The perceptron is often used in a modified form
- A scalar bias value can be added to the output computation:

$$\hat{y} = \operatorname{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{2}$$

- The sign function can be replaced with a generic function f:

$$\hat{y} = f\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{3}$$

- These modified perceptrons are often called *neurons* or simply *units*

Generalizations
- The perceptron is often used in a modified form
- A scalar bias value can be added to the output computation:
$$\hat{y} = \operatorname{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{2}$$
- The sign function can be replaced with a generic function f:
$$\hat{y} = f\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{3}$$
- These modified perceptrons are often called *neurons* or simply *units*

f: Aktivierungsfunktion

Generalizations

- The perceptron is often used in a modified form
- A scalar bias value can be added to the output computation:

$$\hat{y} = \operatorname{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{2}$$

- The sign function can be replaced with a generic function f:

$$\hat{y} = f\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{3}$$

- These modified perceptrons are often called *neurons* or simply *units*
- **Notation**: We denote the *weighted input* as

$$z = \mathbf{w}^\top \mathbf{x} + b \tag{4}$$

Generalizations
- The perceptron is often used in a modified form
- A scalar bias value can be added to the output computation:
$$\hat{y} = \operatorname{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{2}$$
- The sign function can be replaced with a generic function f:
$$\hat{y} = f\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{3}$$
- These modified perceptrons are often called *neurons* or simply *units*
- **Notation**: We denote the *weighted input* as
$$z = \mathbf{w}^\top \mathbf{x} + b \tag{4}$$

f: Aktivierungsfunktion

Feedforward Neural Networks

Feedforward Neural Networks

Perzeptron linear: Viele Funktionen können nicht gelernt werden

Networks of neurons

- **Idea**: A combination of multiple neurons could make much better predictions

Networks of neurons

- **Idea**: A combination of multiple neurons could make much better predictions

Angelehnt an Gehirn

Auch Multilayer Perzeptron

Networks of neurons

- **Idea**: A combination of multiple neurons could make much better predictions
- A feedforward neural network is a layered architecture of neurons

Networks of neurons
- **Idea**: A combination of multiple neurons could make much better predictions
- A feedforward neural network is a layered architecture of neurons

Angelehnt an Gehirn

Auch Multilayer Perzeptron

Visual Representation

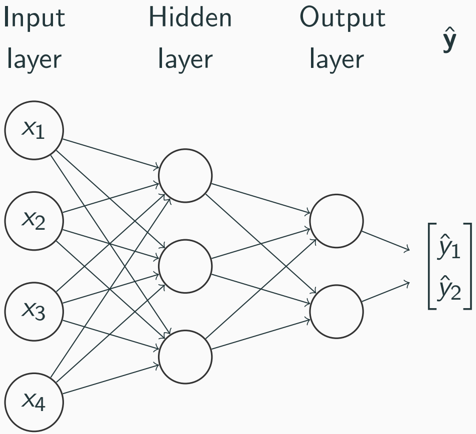
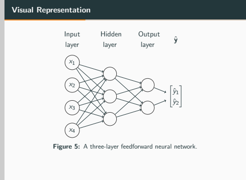


Figure 5: A three-layer feedforward neural network.



Tiefe

Verbindungen nur zwischen Layers

Output Layer

- The design of the output layer depends on the task that we wish to perform

Output Layer

- The design of the output layer depends on the task that we wish to perform

Output Layer

- The design of the output layer depends on the task that we wish to perform
- *Regression*: one single linear neuron

Output Layer

- The design of the output layer depends on the task that we wish to perform
- *Regression*: one single linear neuron

Output Layer

- The design of the output layer depends on the task that we wish to perform
- *Regression*: one single linear neuron
- *Binary classification*: one single sigmoid neuron

Output Layer

- The design of the output layer depends on the task that we wish to perform
- *Regression*: one single linear neuron
- *Binary classification*: one single sigmoid neuron

The Logistic Sigmoid Function

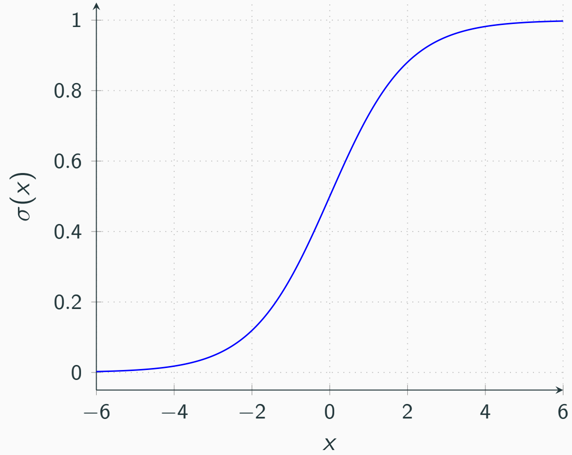
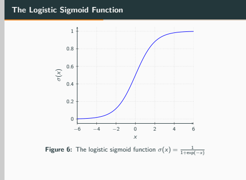


Figure 6: The logistic sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$



Output Layer

- The design of the output layer depends on the task that we wish to perform
- *Regression*: one single linear neuron
- *Binary classification*: one single sigmoid neuron
- *Multiclass classification*: k output units with the softmax function

$$\mathrm{softmax}(x) = \frac{\exp(x)}{\sum_{i=1}^{k} \exp(z_i)} \tag{5}$$

Output Layer
- The design of the output layer depends on the task that we wish to perform
- *Regression*: one single linear neuron
- *Binary classification*: one single sigmoid neuron
- *Multiclass classification*: k output units with the softmax function
$$\mathrm{softmax}(x) = \frac{\exp(x)}{\sum_{i=1}^{k} \exp(z_i)} \tag{5}$$

softmax: Generalisierung, normalisiert

Viele andere Probleme: Auffälliges Verhalten, Bildgenerierung

Hidden Layers

- The task does not give us any information about how to design the hidden layers

Hidden Layers
- The task does not give us any information about how to design the hidden layers

Daher kommt Deep Learning
Verschiedene Repräsentationen (Kanten, Objekte,...) $\Rightarrow$ Abstraktionen!
Kontinuierlich, nicht linear!

Hidden Layers

- The task does not give us any information about how to design the hidden layers
- Depth: Irrelevant from a theoretical point of view

Hidden Layers
- The task does not give us any information about how to design the hidden layers
- Depth: Irrelevant from a theoretical point of view

Daher kommt Deep Learning
Verschiedene Repräsentationen (Kanten, Objekte,...) $\Rightarrow$ Abstraktionen!
Kontinuierlich, nicht linear!

Hidden Layers

- The task does not give us any information about how to design the hidden layers
- Depth: Irrelevant from a theoretical point of view
- Deep networks perform almost always better in practice

Hidden Layers

- The task does not give us any information about how to design the hidden layers
- Depth: Irrelevant from a theoretical point of view
- Deep networks perform almost always better in practice

Daher kommt Deep Learning
Verschiedene Repräsentationen (Kanten, Objekte,...) $\Rightarrow$ Abstraktionen!
Kontinuierlich, nicht linear!

Hidden Layers

- The task does not give us any information about how to design the hidden layers
- Depth: Irrelevant from a theoretical point of view
- Deep networks perform almost always better in practice
- Activation function: Three common choices are the logistic sigmoid, the tanh, and the rectified linear function

Hidden Layers

- The task does not give us any information about how to design the hidden layers
- Depth: Irrelevant from a theoretical point of view
- Deep networks perform almost always better in practice
- Activation function: Three common choices are the logistic sigmoid, the tanh, and the rectified linear function

Daher kommt Deep Learning

Verschiedene Repräsentationen (Kanten, Objekte,...) $\Rightarrow$ Abstraktionen!

Kontinuierlich, nicht linear!

The Rectified Linear Function

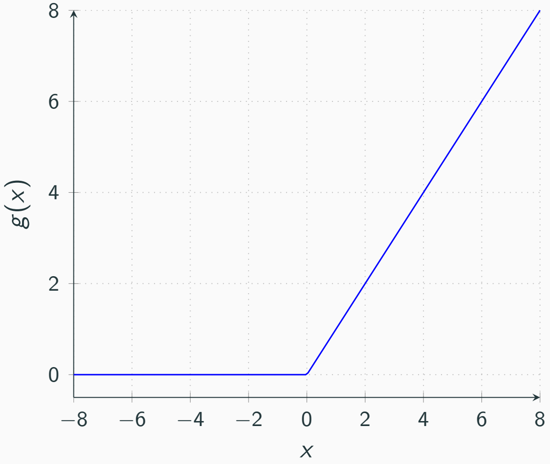
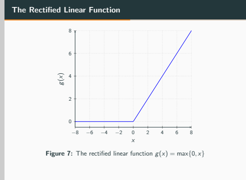


Figure 7: The rectified linear function $g(x) = \max\{0, x\}$



Hidden Layers

- The task does not give us any information about how to design the hidden layers
- Depth: Irrelevant from a theoretical point of view
- Deep networks perform almost always better in practice
- Activation function: Three common choices are the logistic sigmoid, the tanh, and the rectified linear function
- Experimentation and trial & error

Hidden Layers
- The task does not give us any information about how to design the hidden layers
- Depth: Irrelevant from a theoretical point of view
- Deep networks perform almost always better in practice
- Activation function: Three common choices are the logistic sigmoid, the tanh, and the rectified linear function
- Experimentation and trial & error

Mathematical Formulation

- We can specify a single neuron with a weight vector $\mathbf{w}$ and a bias value b

Mathematical Formulation

- We can specify a single neuron with a weight vector $\mathbf{w}$ and a bias value b

f wird komponentenweise angewendet

Mathematical Formulation

- We can specify a single neuron with a weight vector $\mathbf{w}$ and a bias value b
- Since a neural network consists of multiple neurons in a layer, we need weight *matrices* $\mathbf{W}^{(l)}$ and bias *vectors* $\mathbf{b}^{(l)}$ to specify the parameters of a layer l

Mathematical Formulation

- We can specify a single neuron with a weight vector $\mathbf{w}$ and a bias value b
- Since a neural network consists of multiple neurons in a layer, we need weight *matrices* $\mathbf{W}^{(l)}$ and bias *vectors* $\mathbf{b}^{(l)}$ to specify the parameters of a layer l

f wird komponentenweise angewendet

Mathematical Formulation

- We can specify a single neuron with a weight vector $\mathbf{w}$ and a bias value b
- Since a neural network consists of multiple neurons in a layer, we need weight *matrices* $\mathbf{W}^{(l)}$ and bias *vectors* $\mathbf{b}^{(l)}$ to specify the parameters of a layer l
- $f^{(l)}$ is the activation function used in the l^{th} layer

Mathematical Formulation

- We can specify a single neuron with a weight vector $\mathbf{w}$ and a bias value b
- Since a neural network consists of multiple neurons in a layer, we need weight *matrices* $\mathbf{W}^{(l)}$ and bias *vectors* $\mathbf{b}^{(l)}$ to specify the parameters of a layer l
- $f^{(l)}$ is the activation function used in the l^{th} layer

f wird komponentenweise angewendet

Mathematical Formulation

- The output at layer l is then given by

$$\mathbf{a}^{(l)} = f^{(l)}\left(\mathbf{W}^{(l)\top}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right) \tag{6}$$

Mathematical Formulation
- The output at layer l is then given by
$$\mathbf{a}^{(l)} = f^{(l)}\left(\mathbf{W}^{(l)\top}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right) \tag{6}$$

Neue Perspektive

$a^0 = x$

$\hat{y} = a^L$

Training Feedforward Neural Networks

Training Feedforward Neural Networks

Training Scenario

- We have training examples $\mathbb{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})$ with corresponding labels $\mathbb{Y}$

Training Scenario
- We have training examples $\mathbb{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})$ with corresponding labels $\mathbb{Y}$

Zufällig initialisiert

Training Scenario

- We have training examples $\mathbb{X} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)})$ with corresponding labels $\mathbb{Y}$
- We want to learn a mapping from $\mathbb{X}$ to $\mathbb{Y}$

Training Scenario

- We have training examples $\mathbb{X} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)})$ with corresponding labels $\mathbb{Y}$
- We want to learn a mapping from $\mathbb{X}$ to $\mathbb{Y}$

Zufällig initialisiert

Training Scenario

- We have training examples $\mathbb{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})$ with corresponding labels $\mathbb{Y}$
- We want to learn a mapping from $\mathbb{X}$ to $\mathbb{Y}$
- **Idea**: Iteratively adjust the parameters of the neural network

Training Scenario

- We have training examples $\mathbb{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})$ with corresponding labels $\mathbb{Y}$
- We want to learn a mapping from $\mathbb{X}$ to $\mathbb{Y}$
- **Idea**: Iteratively adjust the parameters of the neural network

Zufällig initialisiert

Cost Functions

- The cost function $J(\boldsymbol{\theta})$ is a measure of how good the network performs

Cost Functions
- The cost function $J(\boldsymbol{\theta})$ is a measure of how good the network performs

Von den Parametern zu einem Skalar
Größer als 0
Auch Loss oder Error

Cost Functions

- The cost function $J(\theta)$ is a measure of how good the network performs
- Learning can be framed as minimizing the cost function

Cost Functions
- The cost function $J(\theta)$ is a measure of how good the network performs
- Learning can be framed as minimizing the cost function

Von den Parametern zu einem Skalar

Größer als 0

Auch Loss oder Error

Cost Functions

- The cost function $J(\boldsymbol{\theta})$ is a measure of how good the network performs
- Learning can be framed as minimizing the cost function
- The total cost is a sum over the costs of the individual training examples:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \boldsymbol{\theta}) \tag{7}$$

Cost Functions

- The cost function $J(\boldsymbol{\theta})$ is a measure of how good the network performs
- Learning can be framed as minimizing the cost function
- The total cost is a sum over the costs of the individual training examples:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \boldsymbol{\theta}) \tag{7}$$

Von den Parametern zu einem Skalar

Größer als 0

Auch Loss oder Error

Mean squared error

- In regression, the per-example loss is commonly

$$\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}) = \frac{1}{2}(\hat{y} - y)^2 \tag{8}$$

Mean squared error
- In regression, the per-example loss is commonly
$$\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}) = \frac{1}{2}(\hat{y} - y)^2 \tag{8}$$

Label: Skalar was wir vorhersagen wollen
Distanz
Erfüllt Bedingungen

Cross-entropy

- In binary classification, we often use the cross-entropy loss

$$\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}) = -y \ln \hat{y} - (1 - y) \ln(1 - \hat{y}) \tag{9}$$

Cross-entropy

- In binary classification, we often use the cross-entropy loss

$$\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}) = -y \ln \hat{y} - (1 - y) \ln(1 - \hat{y}) \tag{9}$$

MSE schlecht in Klassifikation
Label: 1 oder 0
– ln aufzeichnen!

Cross-entropy

- In multiclass classification, the cross-entropy becomes

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = -\ln \hat{y}_i \tag{10}$$

Cross-entropy
• In multiclass classification, the cross-entropy becomes
$\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = -\ln \hat{y}_i$ (10)

Label: i-te Klasse

Maximum Likelihood Estimation

Stochastic Gradient Descent

- (Stochastic) Gradient Descent is the most common algorithm to minimize cost functions in neural networks

Stochastic Gradient Descent
- (Stochastic) Gradient Descent is the most common algorithm to minimize cost functions in neural networks

⇒ Kleine Änderungen in die entgegengesetze Richtung des Gradienten
Learning Rate Hyperparameter durch rumexperimentieren
Erweiterungen

Stochastic Gradient Descent

- (Stochastic) Gradient Descent is the most common algorithm to minimize cost functions in neural networks
- A change $\Delta\boldsymbol{\theta}$ in the parameters corresponds roughly to the change

$$\Delta J(\boldsymbol{\theta}) \approx \nabla J(\boldsymbol{\theta})^\top \Delta\boldsymbol{\theta} \tag{11}$$

Stochastic Gradient Descent
- (Stochastic) Gradient Descent is the most common algorithm to minimize cost functions in neural networks
- A change $\Delta\boldsymbol{\theta}$ in the parameters corresponds roughly to the change
$\Delta J(\boldsymbol{\theta}) \approx \nabla J(\boldsymbol{\theta})^\top \Delta\boldsymbol{\theta}$ (11)

$\Rightarrow$ Kleine Änderungen in die entgegengesetze Richtung des Gradienten
Learning Rate Hyperparameter durch rumexperimentieren
Erweiterungen

Stochastic Gradient Descent

- (Stochastic) Gradient Descent is the most common algorithm to minimize cost functions in neural networks
- A change $\Delta\boldsymbol{\theta}$ in the parameters corresponds roughly to the change

$$\Delta J(\boldsymbol{\theta}) \approx \nabla J(\boldsymbol{\theta})^\top \Delta\boldsymbol{\theta} \tag{11}$$

- To minimize $J(\boldsymbol{\theta})$, choose

$$\Delta\boldsymbol{\theta} = -\eta \nabla J(\boldsymbol{\theta}), \tag{12}$$

Stochastic Gradient Descent

- (Stochastic) Gradient Descent is the most common algorithm to minimize cost functions in neural networks
- A change $\Delta\boldsymbol{\theta}$ in the parameters corresponds roughly to the change $\Delta J(\boldsymbol{\theta}) \approx \nabla J(\boldsymbol{\theta})^\top \Delta\boldsymbol{\theta}$ (11)
- To minimize $J(\boldsymbol{\theta})$, choose $\Delta\boldsymbol{\theta} = -\eta \nabla J(\boldsymbol{\theta})$, (12)

$\Rightarrow$ Kleine Änderungen in die entgegengesetze Richtung des Gradienten
Learning Rate Hyperparameter durch rumexperimentieren
Erweiterungen

Stochastic Gradient Descent

- (Stochastic) Gradient Descent is the most common algorithm to minimize cost functions in neural networks
- A change $\Delta\boldsymbol{\theta}$ in the parameters corresponds roughly to the change

$$\Delta J(\boldsymbol{\theta}) \approx \nabla J(\boldsymbol{\theta})^\top \Delta\boldsymbol{\theta} \tag{11}$$

- To minimize $J(\boldsymbol{\theta})$, choose

$$\Delta\boldsymbol{\theta} = -\eta \nabla J(\boldsymbol{\theta}), \tag{12}$$

- *Stochastic* Gradient Descent computes only an approximation of the gradient

Stochastic Gradient Descent

- (Stochastic) Gradient Descent is the most common algorithm to minimize cost functions in neural networks
- A change $\Delta\boldsymbol{\theta}$ in the parameters corresponds roughly to the change

$$\Delta J(\boldsymbol{\theta}) \approx \nabla J(\boldsymbol{\theta})^\top \Delta\boldsymbol{\theta} \tag{11}$$

- To minimize $J(\boldsymbol{\theta})$, choose

$$\Delta\boldsymbol{\theta} = -\eta \nabla J(\boldsymbol{\theta}), \tag{12}$$

- *Stochastic* Gradient Descent computes only an approximation of the gradient

⇒ Kleine Änderungen in die entgegengesetze Richtung des Gradienten
Learning Rate Hyperparameter durch rumexperimentieren

Erweiterungen

Stochastic Gradient Descent

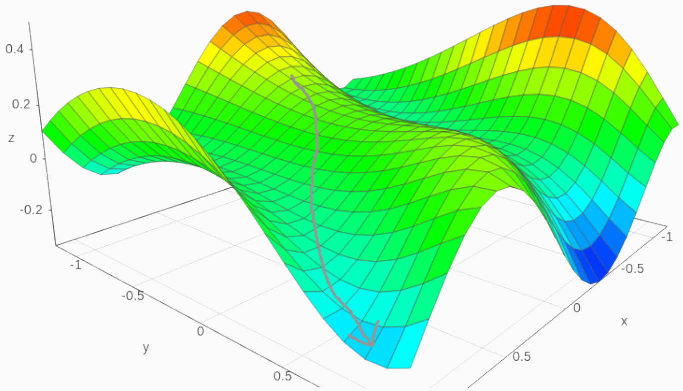
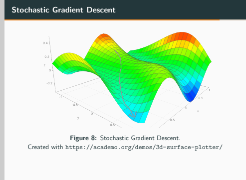


Figure 8: Stochastic Gradient Descent.
Created with `https://academo.org/demos/3d-surface-plotter/`



Back-propagation

- The back-propagation algorithm efficiently computes the gradient of the cost function

Back-propagation
- The back-propagation algorithm efficiently computes the gradient of the cost function

Back-propagation

- The back-propagation algorithm efficiently computes the gradient of the cost function
- It can be derived by recursively applying the chain rule to the layers of the neural network, beginning with the output layer

Back-propagation

- The back-propagation algorithm efficiently computes the gradient of the cost function
- It can be derived by recursively applying the chain rule to the layers of the neural network, beginning with the output layer

Back-propagation

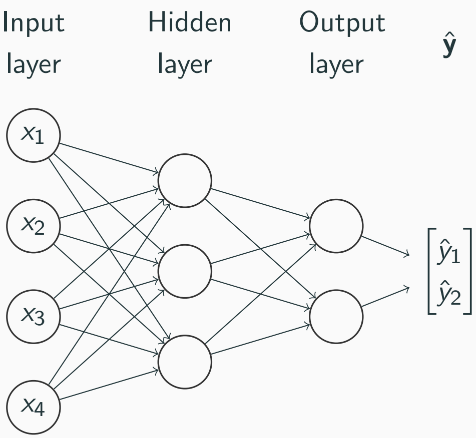
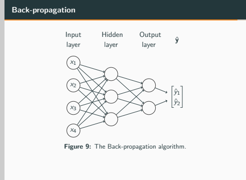


Figure 9: The Back-propagation algorithm.



Back-propagation

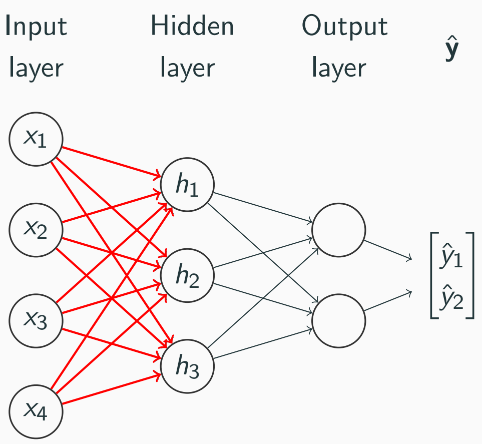
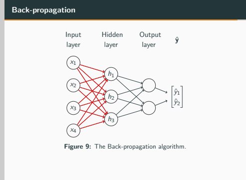


Figure 9: The Back-propagation algorithm.



Back-propagation

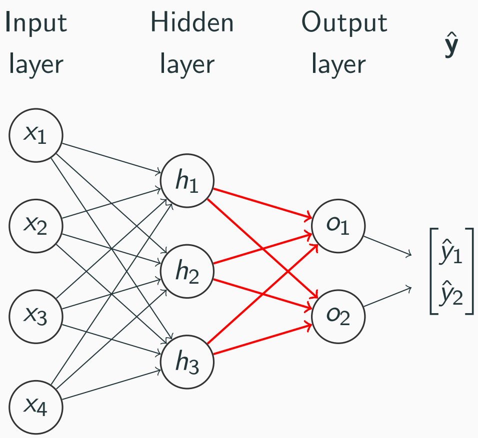
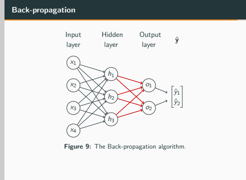


Figure 9: The Back-propagation algorithm.



Back-propagation

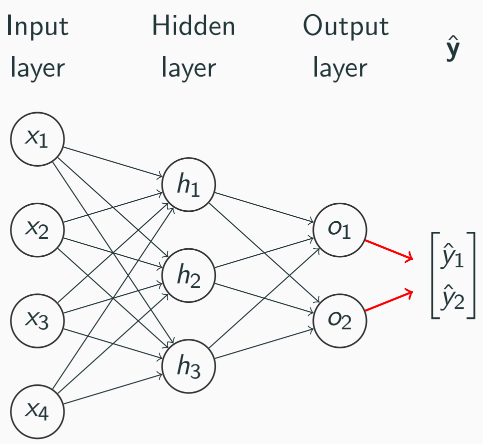
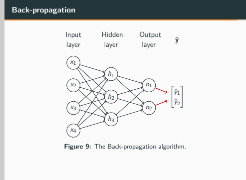


Figure 9: The Back-propagation algorithm.



Back-propagation

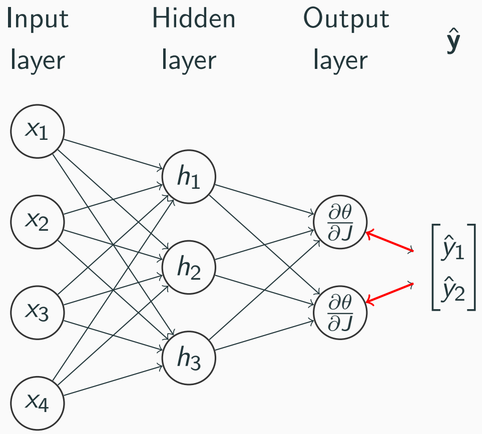
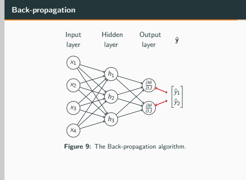


Figure 9: The Back-propagation algorithm.



Back-propagation

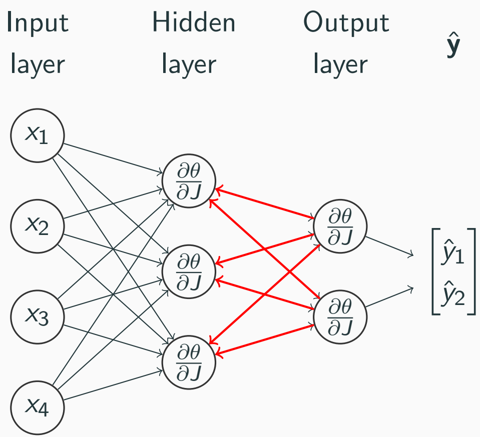
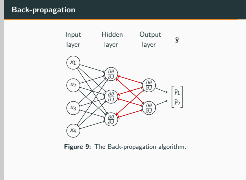


Figure 9: The Back-propagation algorithm.


Back-propagation
Figure 9: The Back-propagation algorithm.

Back-propagation

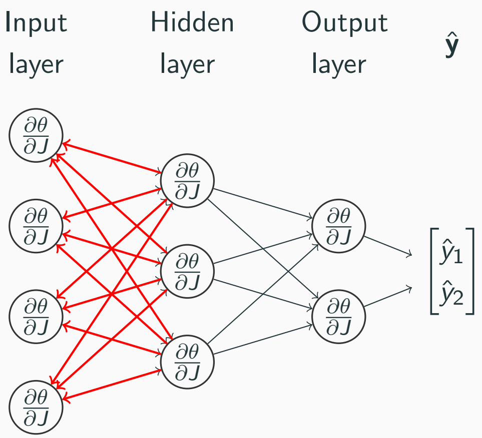
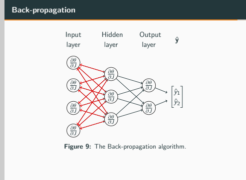


Figure 9: The Back-propagation algorithm.



Back-propagation

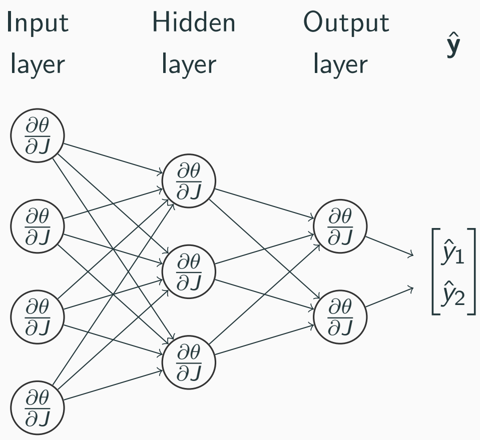
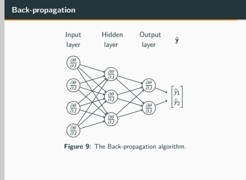


Figure 9: The Back-propagation algorithm.



The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network

The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network

The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network
2. Compute the cost for each training example

The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network
2. Compute the cost for each training example

The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network
2. Compute the cost for each training example
3. Compute all gradients using back-propagation

The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network
2. Compute the cost for each training example
3. Compute all gradients using back-propagation

The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network
2. Compute the cost for each training example
3. Compute all gradients using back-propagation
4. Compute the average gradient

The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network
2. Compute the cost for each training example
3. Compute all gradients using back-propagation
4. Compute the average gradient

The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network
2. Compute the cost for each training example
3. Compute all gradients using back-propagation
4. Compute the average gradient
5. Update the parameters in the negative direction of the gradient

The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network
2. Compute the cost for each training example
3. Compute all gradients using back-propagation
4. Compute the average gradient
5. Update the parameters in the negative direction of the gradient

The Complete Learning Algorithm

1. Propagate all training examples of a minibatch forward through the network
2. Compute the cost for each training example
3. Compute all gradients using back-propagation
4. Compute the average gradient
5. Update the parameters in the negative direction of the gradient
6. Repeat until the cost is low enough

The Complete Learning Algorithm
1. Propagate all training examples of a minibatch forward through the network
2. Compute the cost for each training example
3. Compute all gradients using back-propagation
4. Compute the average gradient
5. Update the parameters in the negative direction of the gradient
6. Repeat until the cost is low enough

Conclusion

Conclusion

Komplexe Netzwerke einfacher Einheiten
Abstraktionen
Lernen: Kleine Updates der Parameter so dass das Netzwerk besser wird
Überall in Deep Learning

Viele weitere Anwendungen in der Zukunft

Thank you!

Thank you!

Fragen?