Wie vollbringen künstliche Intelligenzen das Kunststück des Lernens?

Ingo Blechschmidt

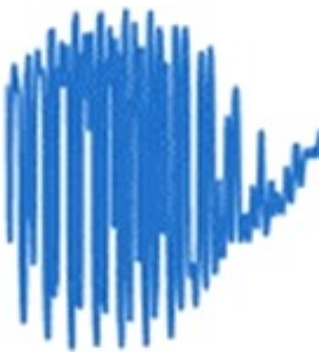
mit Dank an Tim Baumann und Philipp Wacker

Institut für Mathematik der Universität Augsburg

16. Augsburger Linux-Infotag am 22. April 2017

Teil I

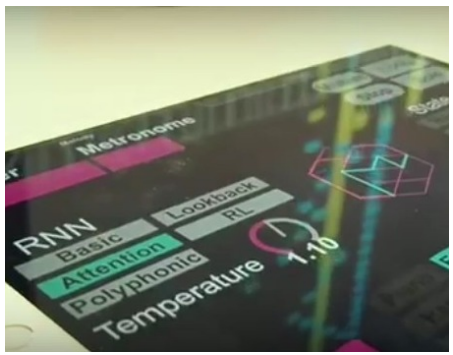
Jüngste Erfolge von künstlicher Intelligenz



Sprachsynthese

AlphaGo

Stiltransfer



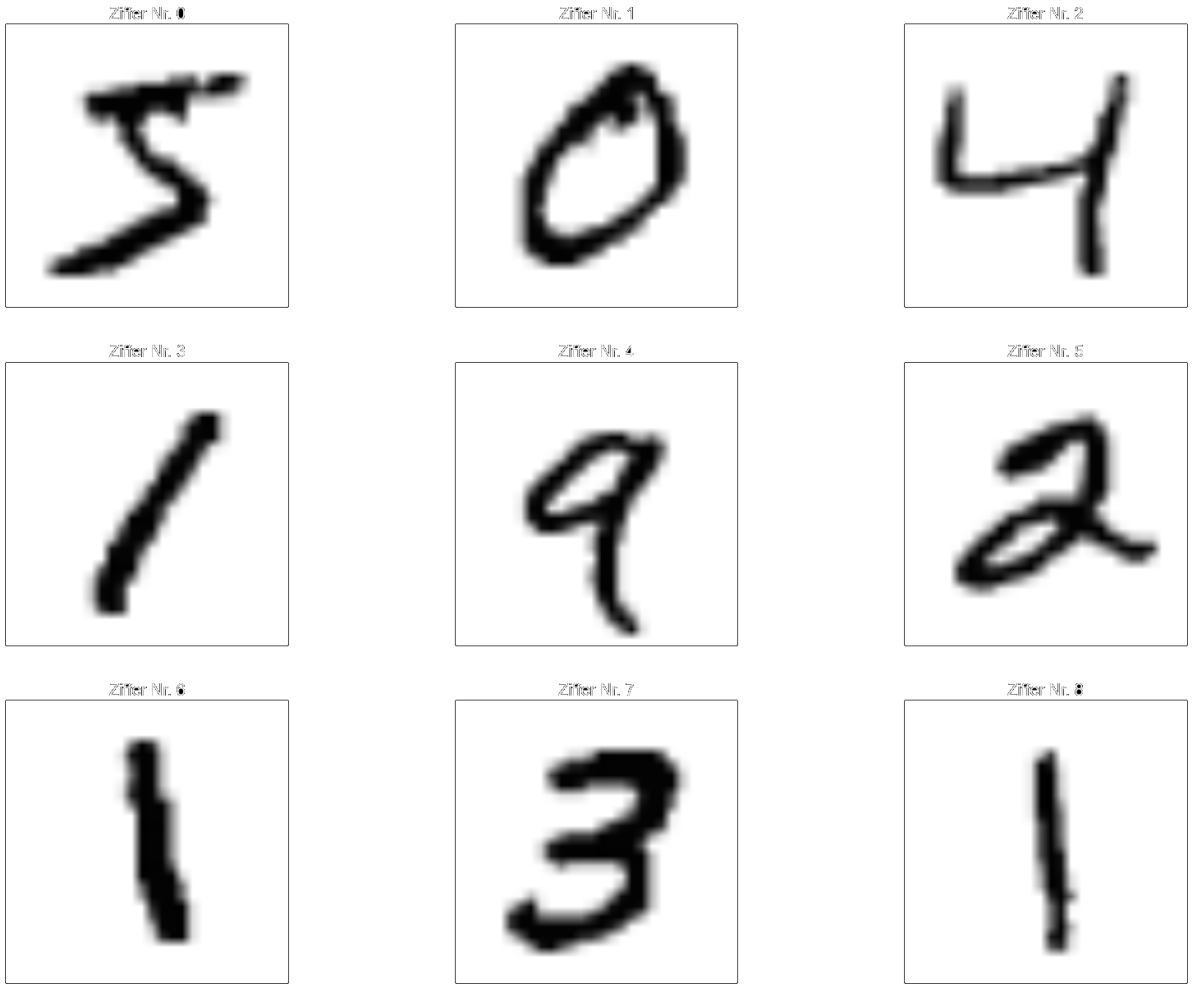
Jammen mit Magenta

Teil II

Funktionsweise künstlicher neuronaler Netzwerke

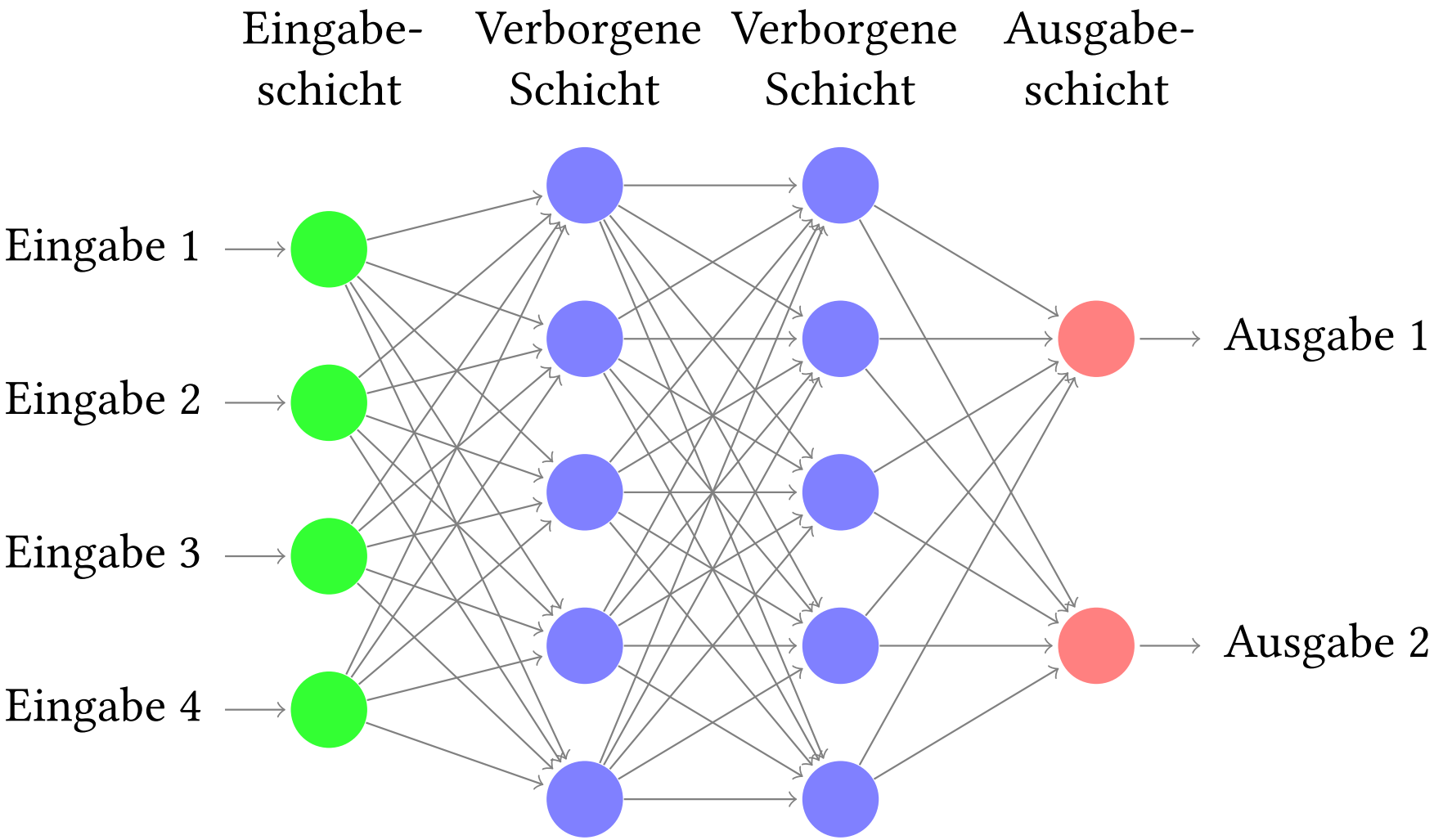
1. Aufbau eines einfachen Netzes
2. Bewertung durch eine Kostenfunktion
3. Fehlerminimierung mittels Gradientenabstieg

Der MNIST-Datensatz

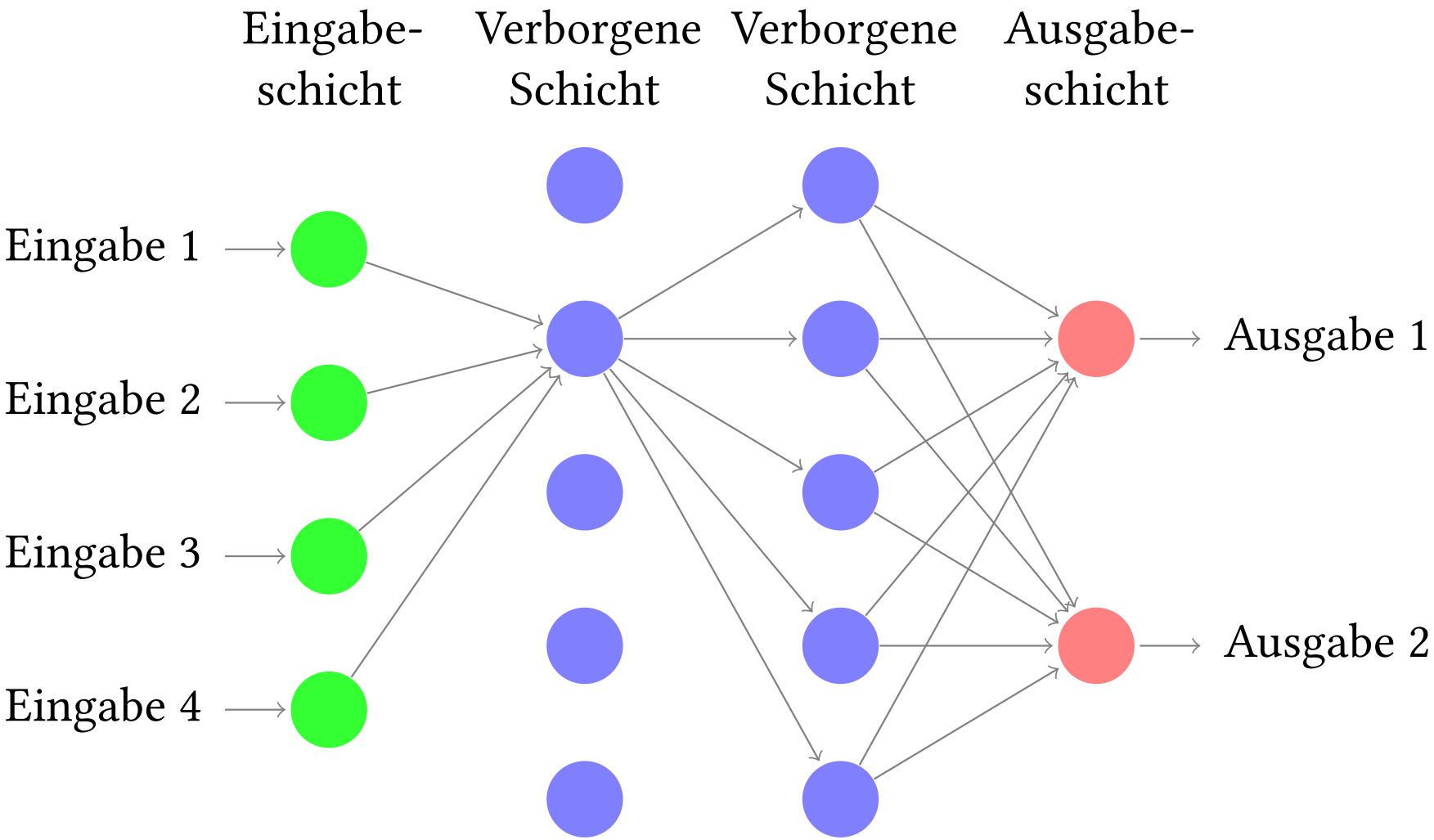


70 000 Bilder mit je 28×28 Pixeln

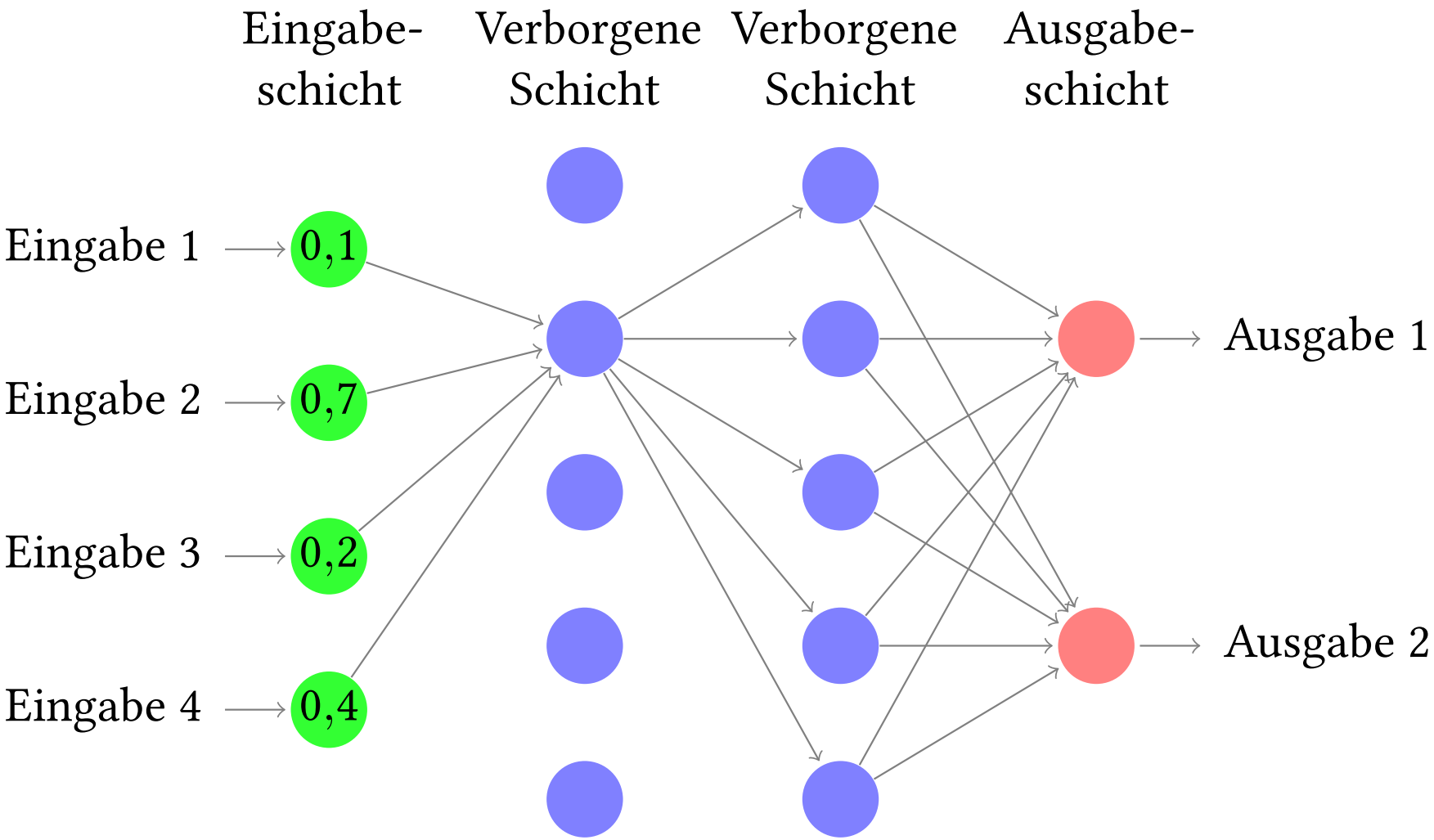
Aufbau eines einfachen Netzes



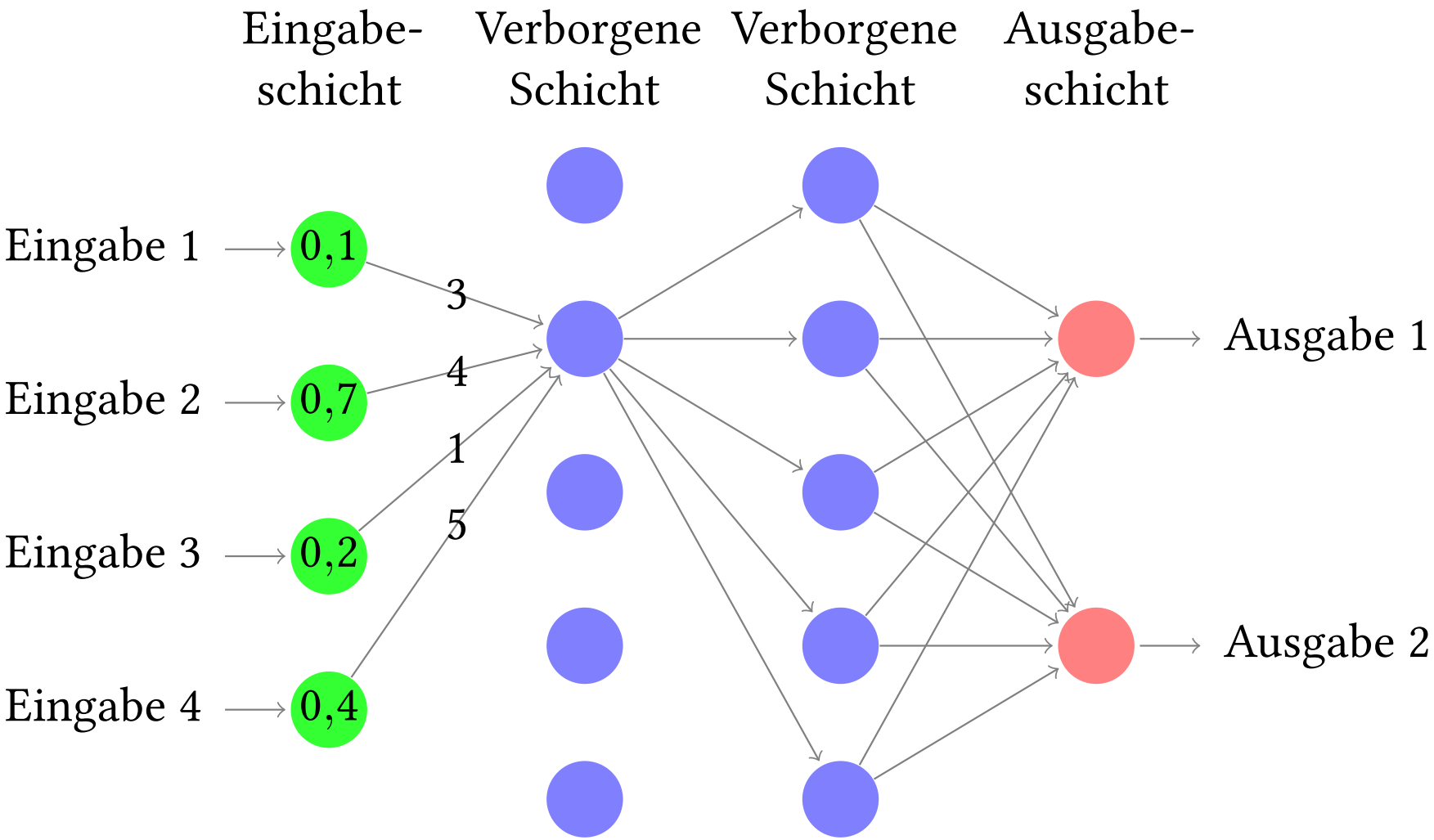
Aufbau eines einfachen Netzes



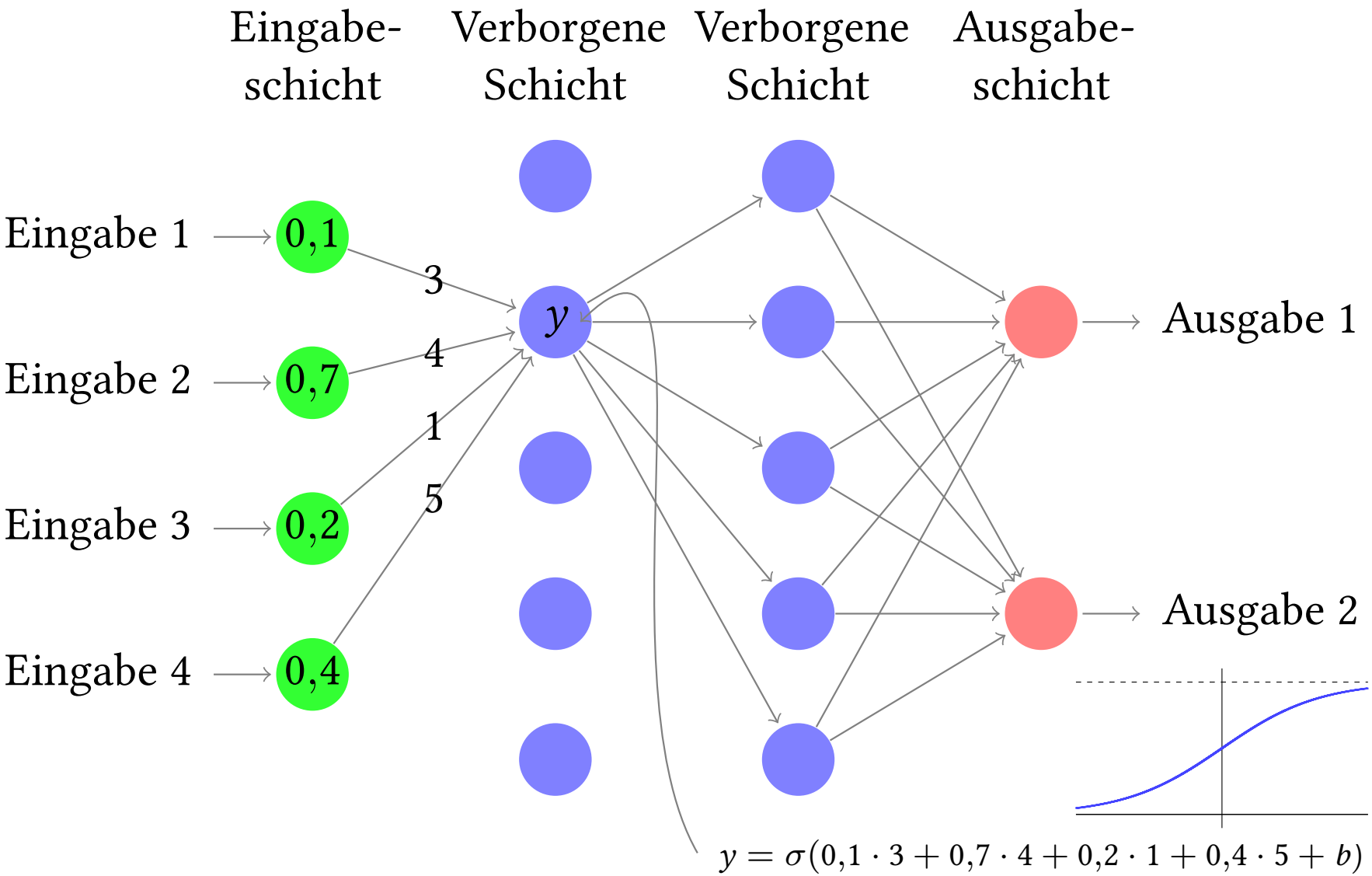
Aufbau eines einfachen Netzes



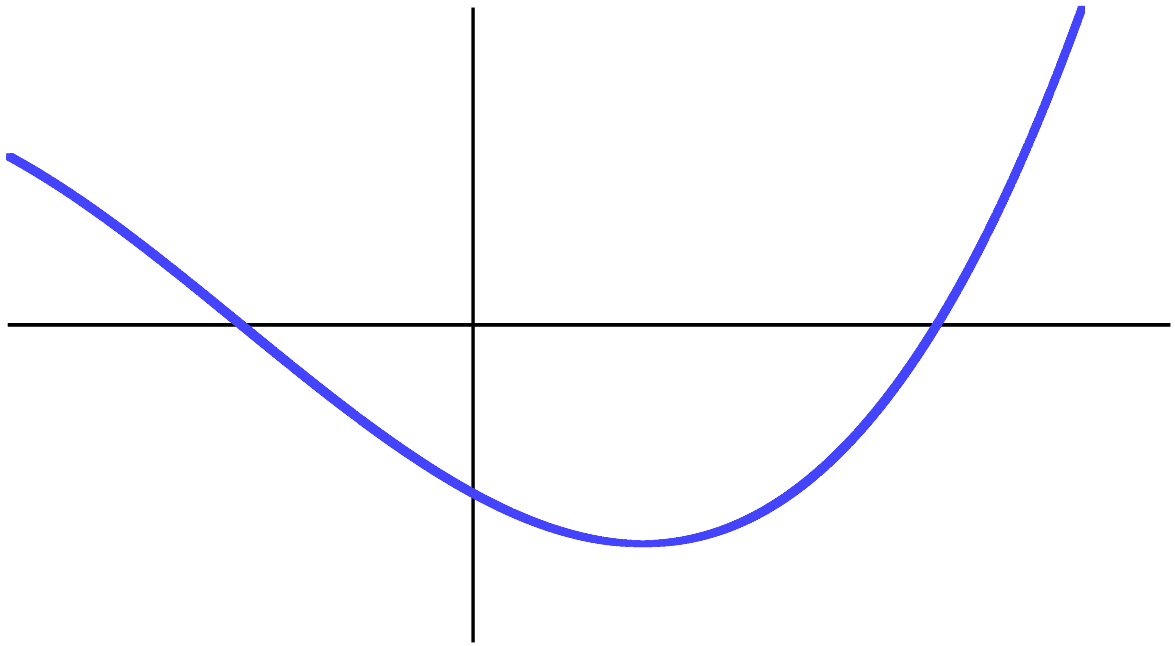
Aufbau eines einfachen Netzes



Aufbau eines einfachen Netzes

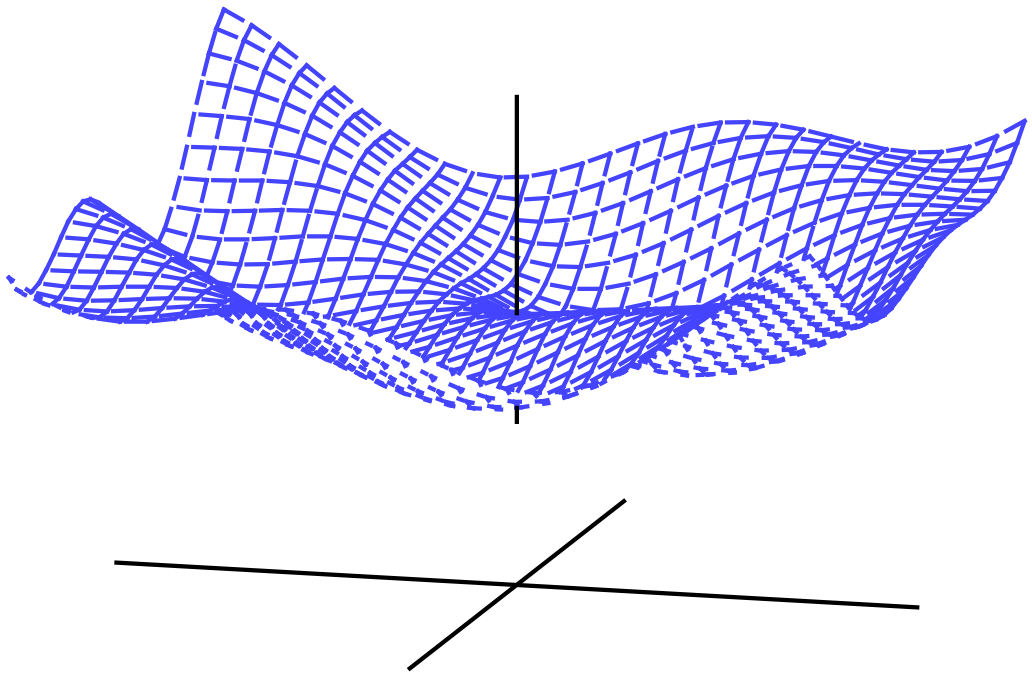


Das Kunststück des Lernens



eine Unbekannte: x

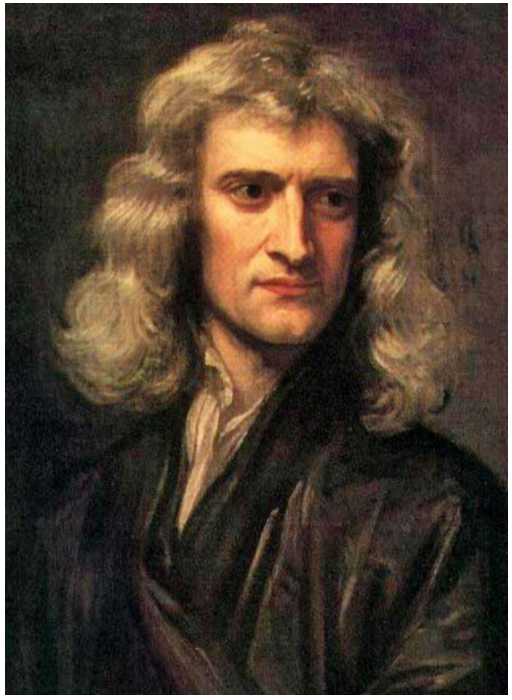
Das Kunststück des Lernens



zwei Unbekannte: x, y

Das Kunststück des Lernens

Leibniz (* 1646, † 1716)



Newton (* 1643, † 1727)

beliebig viele Unbekannte

Das Kunststück des Lernens

1. Bestimme für jeden der 60 000 Trainingsfälle die Aktivierungen der zehn Ausgabeneuronen.
2. Summiere für jede der resultierenden 600 000 Aktivierungen die einzelnen **quadratischen Fehler** auf, um die **Gesamtkosten** zu bestimmen:

$$\underbrace{(0.1-0)^2+(0.7-\mathbf{1})^2+(0.1-0)^2+\cdots+(0.2-0)^2}_{\text{erster Trainingsfall (sollte als 1 erkannt werden)}}$$
$$+\underbrace{(0.3-\mathbf{1})^2+(0.2-0)^2+(0.2-0)^2+\cdots+(0.1-0)^2}_{\text{zweiter Trainingsfall (sollte als 0 erkannt werden)}}$$
$$+\cdots$$

3. Ändere die Gewichte und Verschiebungen ein kleines bisschen in Richtung des **steilsten Abstiegs**, um die Gesamtkosten ein kleines bisschen zu reduzieren, das Netzwerk also ein kleines bisschen zu verbessern.
4. Gehe zu Schritt 1.

Im MNIST-Beispiel legen wir von den 70 000 Einträgen 10 000 beiseite; diese verwenden wir später zur Validierung. Den Lernvorgang beginnen wir mit einer rein zufälligen Wahl von Gewichten und Biases.

1. Wir berechnen für jede der 60 000 Trainingsfälle die Aktivierungen der zehn Ausgabeneuronen. So erhalten wir insgesamt 600 000 Zahlen zwischen 0 und 1. Für jedes dieser Ergebnisse wissen wir, welchen Wert wir uns eigentlich wünschen (jeweils 0 oder 1 – etwa soll Ausgabeneuron Nr. 5 bei Eingabe einer handschriftlichen Sieben idealerweise überhaupt nicht feuern).

2. Für jede dieser 600 000 Ergebnisse berechnen wir den *quadratischen Fehler*

   $$(\text{tatsächliches Ergebnis} - \text{Wunschergebnis})^2$$

   und summieren all diese Quadrate auf. (Interessiert dich, wieso man hier quadriert? Schreibe eine Mail an iblech@speicherleck.de.)

3. Je größer diese Summe ist, desto schlechter funktioniert das Netzwerk auf den Trainingsdaten. Wir möchten daher die Summe *minimieren*. Die Summe hängt von den Gewichten der künstlichen Synapsen und den Biases der Neuronen ab; diese Abhängigkeit heißt auch *Kostenfunktion*.

4. Durch Bestimmung des *Gradienten* wissen wir, wie wir die Gewichte und Biases ändern müssen, um eine kleine Reduktion der Kostenfunktion zu erreichen. So erhalten wir neue Gewichte und Biases. Anschließend beginnen wir wieder bei Schritt 1. Auf diese Weise folgen wir zu jedem Zeitpunkt der Richtung des steilsten Abstiegs im hochdimensionalen Kostengebirge.

Sobald wir mit der Leistung des Netzes auf den Validierungsdatensätzen zufrieden sind, beenden wir das Training. Das *Wunder der Generalisierung* setzt ein: Das Netz klassifiziert auch neu geschriebene Ziffern, die nicht Teil des Trainingdatensatzes waren, sehr häufig richtig.

Im MNIST-Beispiel legen wir von den 70 000 Einträgen 10 000 beiseite; diese verwenden wir später zur Validierung. Den Lernvorgang beginnen wir mit einer rein zufälligen Wahl von Gewichten und Biases.

1. Wir berechnen für jede der 60 000 Trainingsfälle die Aktivierungen der zehn Ausgabeneuronen. So erhalten wir insgesamt 600 000 Zahlen zwischen 0 und 1. Für jedes dieser Ergebnisse wissen wir, welchen Wert wir uns eigentlich wünschen (jeweils 0 oder 1 – etwa soll Ausgabeneuron Nr. 5 bei Eingabe einer handschriftlichen Sieben idealerweise überhaupt nicht feuern).

2. Für jede dieser 600 000 Ergebnisse berechnen wir den *quadratischen Fehler*

   $$(\text{tatsächliches Ergebnis} - \text{Wunschergebnis})^2$$

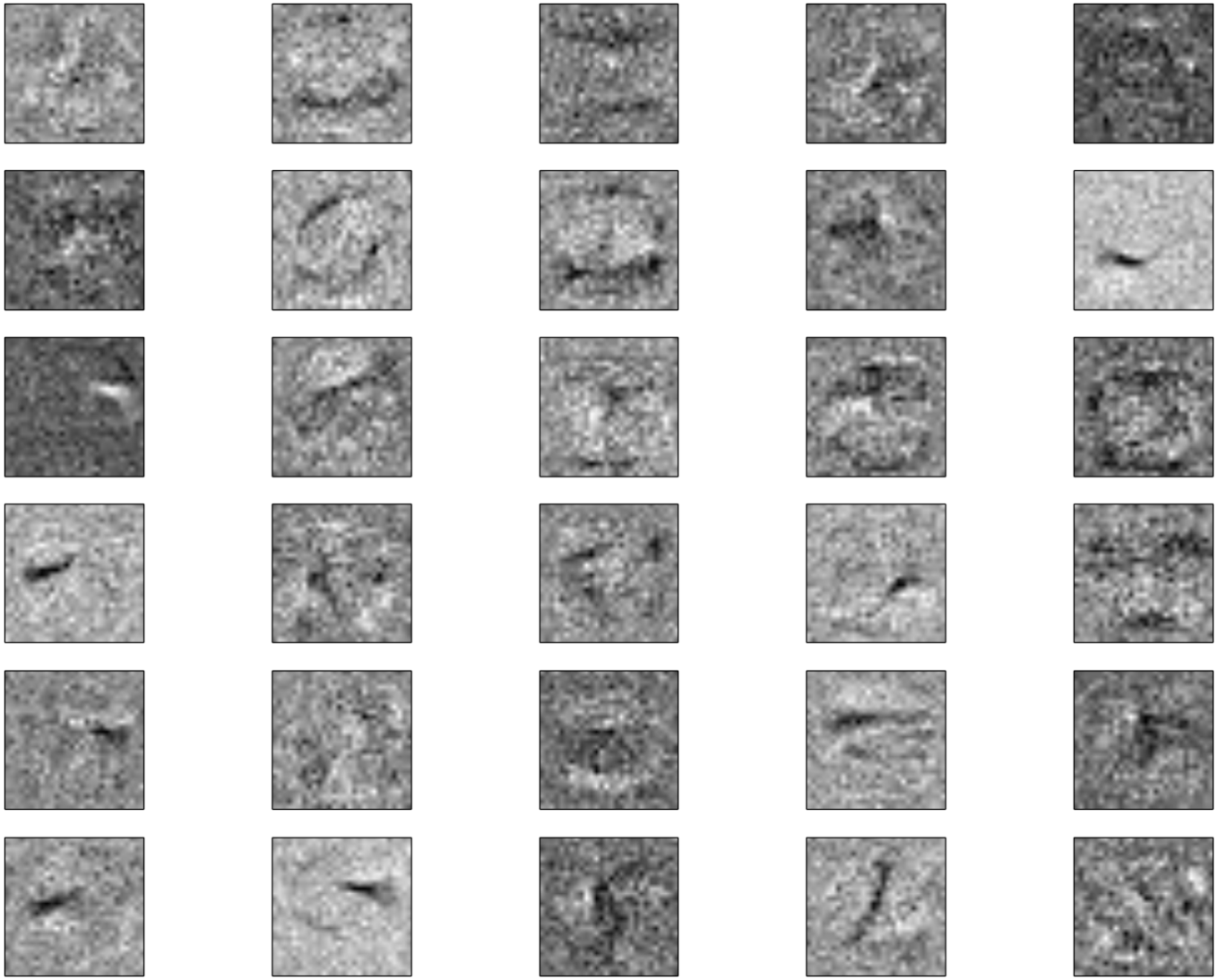
   und summieren all diese Quadrate auf. (Interessiert dich, wieso man hier quadriert? Schreibe eine Mail an iblech@speicherleck.de.)

3. Je größer diese Summe ist, desto schlechter funktioniert das Netzwerk auf den Trainingsdaten. Wir möchten daher die Summe *minimieren*. Die Summe hängt von den Gewichten der künstlichen Synapsen und den Biases der Neuronen ab; diese Abhängigkeit heißt auch *Kostenfunktion*.

4. Durch Bestimmung des *Gradienten* wissen wir, wie wir die Gewichte und Biases ändern müssen, um eine kleine Reduktion der Kostenfunktion zu erreichen. So erhalten wir neue Gewichte und Biases. Anschließend beginnen wir wieder bei Schritt 1. Auf diese Weise folgen wir zu jedem Zeitpunkt der Richtung des steilsten Abstiegs im hochdimensionalen Kostengebirge.

Sobald wir mit der Leistung des Netzes auf den Validierungsdatensätzen zufrieden sind, beenden wir das Training. Das *Wunder der Generalisierung* setzt ein: Das Netz klassifiziert auch neu geschriebene Ziffern, die nicht Teil des Trainingdatensatzes waren, sehr häufig richtig.

Blick in die verborgene Schicht



Hier wurde ein einfaches Netz zur Ziffernerkennung bestehend aus nur einer einzigen verborgenen Schicht mit 30 Neuronen trainiert. Die Grafik zeigt die Gewichte der Synapsen zwischen den 28×28 Eingabeneuronen und diesen 30 Neuronen. Das Netz hat eine Erkennungsrate von 95 %.

Verwendet man 100 Neuronen, so erreicht man 97 %; das ist fast eine Halbierung der Fehlerrate.

Demo zum Selbstprobieren:

- Python-Code zur Erkennung
- Python-Code zum Training

Teil III

Wieso nicht schon früher?

1. Größere Rechenleistung

Teil III

Wieso nicht schon früher?

1. Größere Rechenleistung
2. Verfügbarkeit großer Datensätze zum Training

Teil III

Wieso nicht schon früher?

1. Größere Rechenleistung
2. Verfügbarkeit großer Datensätze zum Training
3. Mathematischer Durchbruch: Convolutional Neural Networks

Teil IV

Herausforderungen für die Zukunft

- Neuronale Netze auf weitere Problemklassen ausdehnen
- Innere Funktionsweise von Netzen verstehen
- Resistenz gegen Adversarial Examples entwickeln
- Ethische Fragen diskutieren
- Existenzielle Fragen bei starker KI untersuchen

Wie ein künstliches neuronales Netzwerk funktioniert, ist – anders als bei herkömmlichem Programmcode – nicht klar. (Wie beim Menschen auch.) Dazu wird momentan aktiv geforscht. Zwei Einstiegspunkte zu solchen Untersuchungen sind:

- Inceptionism: Going Deeper into Neural Networks von Alexander Mordvintsev, Christopher Olah und Mike Tyka
- Visualizing and Understanding Convolutional Networks von Matthew Zeiler und Rob Fergus

Teil V

Empfehlungen

- HBO-Serie Westworld über Androiden, die schon lange den Turingtest bestehen und nun ein Bewusstsein entwickeln
- Vorträge von Joscha Bach auf dem Kongress
- The Unreasonable Effectiveness of Recurrent Neural Networks von Andrej Karpathy
- TensorFlow – mische auch ohne viel Programmiererfahrung bei der KI-Entwicklung mit!
- Neural Networks and Deep Learning von Michael Nielsen