EXXETA
CONSULTING AND TECHNOLOGIES

Developer Open Space
Continuous Improvement

EINFÜHRUNG IN DEEP LEARNING MIT KERAS

Michael Aydinbas

16.12.2021

Michael Aydinbas

Senior Data Engineer

Michael Aydinbas verfügt über 2 Jahre Berufserfahrung als Experte für Data Science und Data Engineering sowie über 10 Jahre Erfahrung mit Softwareentwicklung. Sein Schwerpunkt liegt auf skalierbaren, performanten Big Data Anwendungen. Er hat bereits zahlreiche Projekte im Bereich Automotive begleitet und umgesetzt. Als Data Strategist und Requirements Engineer konzipiert und gestaltet er Messdatenanalyse-Pipelines – von der Datenaufbereitung über deren Analyse bis zur Auslieferung.

Biographie

- EXXETA
- M.Sc. Psychologie in IT, TU Darmstadt
- B.Sc. Umwelting.-Wiss. sowie in Psychologie in IT, TU Darmstadt

Beratungskompetenz

- Data Engineer, Data Scientist, Data Strategist; Requirements Engineer, Datenstrategie und Systemlandschaft
- Schulungen zu Python, Machine Learning und Deep Learning
- Forschung und Entwicklung, Automotive
- Technische Expertise u.a. in Data Science, Large-Scale Data Processing, System Integration

Sprachen

- Englisch, Deutsch

Auszug relevante Projekterfahrung

Projektleiter/Senior Consultant, Big Data im Entwicklungsprozess, Automotive
- Begleitung und Umsetzung des Deployments von entwickelten Analysetools auf eine Big Data Analyse-Plattform
- Entwicklung, Kundenkommunikation, Projektplanung, Dokumentation

Consultant/Requirements Engineer, Big Data im Entwicklungsprozess, Automotive
- Anforderungsanalysen sowie Identifikation und Bewertung von Big Data Use-Cases
- Fachliche Machbarkeit anhand eines Prototyps in der Cloud (AWS) demonstrieren
- Umsetzungen von Visualisierungen und Dashboards mit ElasticSearch + Kibana

Data Scientist/Requirements Engineer, Entwicklung eines Sequencing-Frameworks, Automotive
- Anforderungsanalysen
- Entwicklung einer Python Toolbox für Datenexploration / Datenanalyse mit Apache Spark
- Entwicklung eines Python Sequencing Frameworks zur automatisierten Erkennung von Fahrszenen

Consultant/Requirements Engineer, Entwicklung einer Datenstrategie, Personentransport
- Durchführung einer Standortanalyse und Anforderungsaufnahme durch Interviews mit allen Stakeholdern
- Planung und Durchführung von Experten-Workshops
- Konzeption einer modernen ETL-Strecke mit geeigneten Technologien

AGENDA

- **Tag 1**
  - Einführung Deep Learning
  - Theorie Part I
    - Einfache Netze mit Keras aufbauen und anlernen
- **Tag 2**
  - Theorie Part II
    - Backprop, learning rate & optimizer
    - Modelle evaluieren und anpassen
  - Weiterführende Beispiele: Klassifikation und Regression mit Keras
- **Tag 3**
  - Convolutional Neural Networks (CNN)
  - Transfer Learning
  - Autoencoder
- **Tag 4**
  - Recurrent Neural Networks (RNN)
  - Sequenzdaten mit LSTM analysieren
  - Textdaten mit Embedding-Layers interpretieren

Inhalt

Wie lernen Neuronale Netze?

- Variablen, Modelle und Zusammenhänge
- Künstliche Neuronen und neuronale Netze
- Mathematische Grundlagen des Anlernprozesses: Backpropagation, Loss und Gradient Descent

Neuronale Netze mit Keras aufbauen und trainieren

- Graphen mit Tensorflow erstellen
- Ein einfaches Netz mit Keras und Tensorflow aufbauen und anlernen
- Welches Modell ist das richtige? Evaluation und Anpassung angelernter Modelle
- Einsatz angelernter Modelle zur Klassifzierung und Schätzung
- Modelle speichern und laden
- Overfitting mit Dropout-Layern verhindern
- Überwachung des Anlernprozesses mit TensorBoard

Maschinelles Sehen: Konvolutionale neuronale Netze (CNN)

- Bilddateien als mehrdimensionale Matrizen
- Details sehen und Formen abstrahieren: Konvolutionale Netze (CNN)
- Konvolutionale Layer mit Keras aufbauen und anlernen
- Vortrainierte Layer einsetzen

Mit Sequenzdaten arbeiten: Rekurrente neuronale Netze (RNN)

- Zeitreihendaten und Textsequenzen: Eigenschaften und Eigenheiten
- Neuronale Netze mit Gedächtnis (Feedforward vs. RNN)
- Sequenzdaten mit LSTM (Long Short-Term Memory) aufbauen
- Textdaten mit Embedding-Layers interpretieren

Feature Extraction: Numerische Daten, Bild und Text-Daten vorbereiten

- Datenrepräsentation: Skalare, Vektoren & Matrizen in Numpy
- Daten einlesen und vorbereiten mit Pandas, Scikit-learn und Keras
- Qualitativer Variablen und Dummy-Sets
- Standardisierung von Daten
- Dimensionsreduktion mit PCA
- Bilddateien transformieren
- Mit Textdaten arbeiten: Zerlegen, tokenizieren, vektorisieren und einbetten.

Beschreibung

Neuronale Netze sind die Arbeitstiere der datengetriebenen Ökonomie. Sie automatisieren Prozesse, indem sie Ihre Daten für intelligente Entscheidungen nutzen. Das Seminar gibt eine Einführung in die Techniken des Anlernens und in den Einsatz neuronaler Netze mit *Python* und *Keras*. Sie lernen Bild-, Text- und numerische Daten aufzubereiten, geeignete Modelle anzulernen und zur Klassifikation und Schätzung in der Produktion einzusetzen.

Schulungsziel

Sie lernen den kompletten Workflow des Machine Learnings mit der Deep-Learning Bibliothek Keras kennen. Angefangen von der Datenaufbereitung, über die Auswahl und das Anlernen der Modelle, bis hin zum Einsatz der trainierten Modelle zur Klassifikation und Prognose. Sie können Bild-, Text- und numerische Daten für Training und Schätzung vorbereiten. Sie kennen die konzeptuellen Grundlagen des Anlernens, die *Verfahren* zur Qualitätssicherung im Anlernprozess und die Varianten neuronaler Netze. Dazu gehören konvolutationale Netze (CNN) zur Verarbeitung von Bilddaten genauso wie rekurrente Netze (RNN) zur Verarbeitung von Text- und Zeitreihendaten. Die Konzepte werden anhand von Folien erklärt, an Beispielen verdeutlicht und gemeinsam in Python umgesetzt und eingeübt. Eine Vielzahl praktischer Übungen ergänzen den Lernprozess und ermöglichen es, die verschiedenen Lernszenarien selbstständig umzusetzen. Am Ende des Seminars sind Sie in der Lage, Daten und Optionen des Einsatzes neuronaler Netze zu prüfen, Modelle zielsicher auszuwählen und zu trainieren und Klassifizierer erfolgreich einzusetzen.

Vorstellungsrunde

- Wie heißt Ihr?
- Was habt Ihr für einen Hintergrund?
- Was führt Euch hierher?
- Warum interessiert Euch für Machine Learning und Deep Learning?

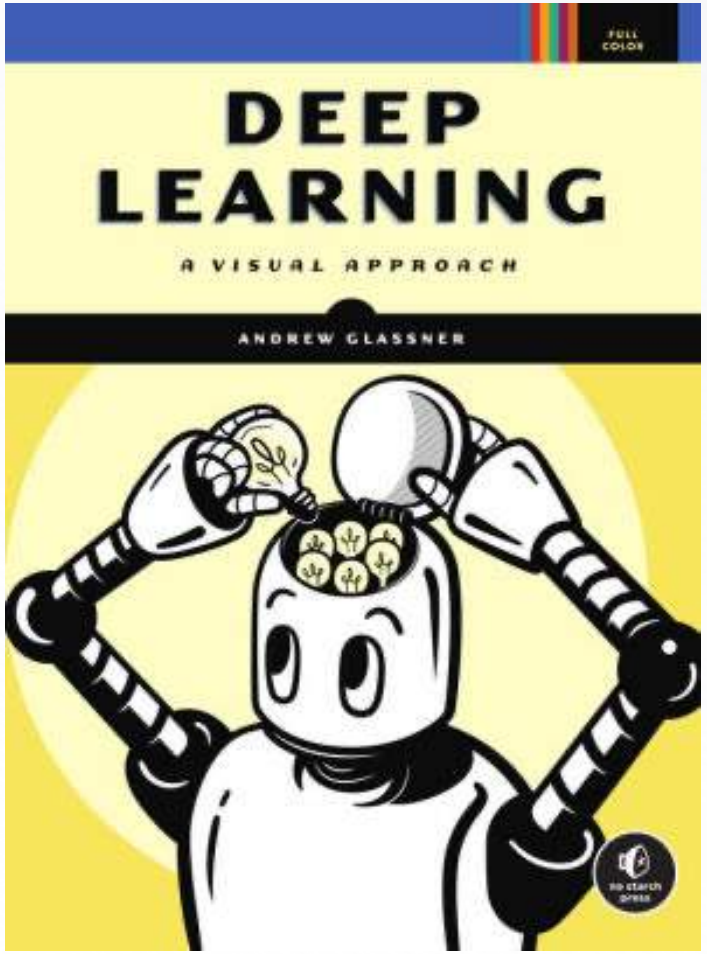
Theorie und Praxis

Material

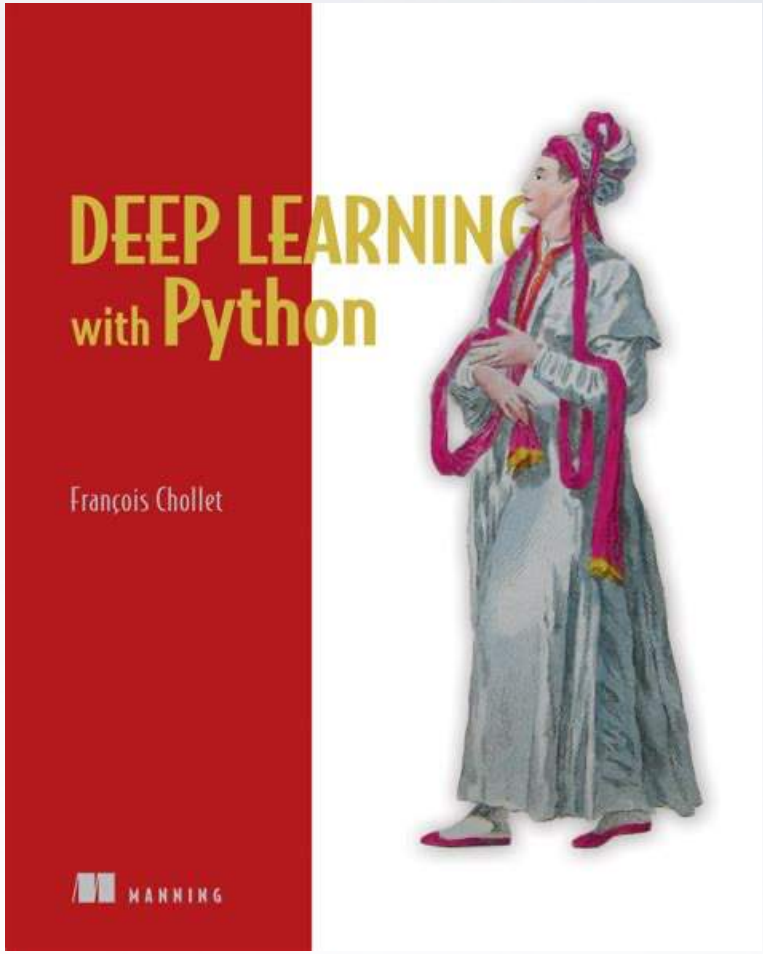
1

Introduction to deep learning

Literature for this course



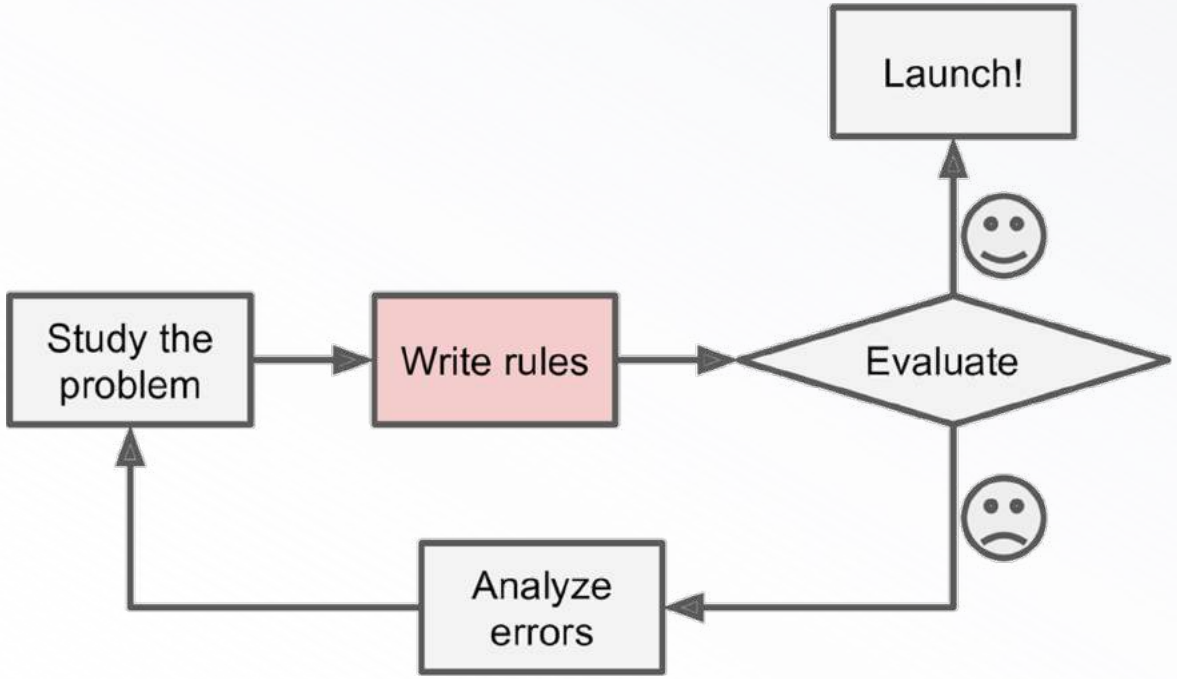
Deep Learning: A Visual Approach
by Andrew Glassner

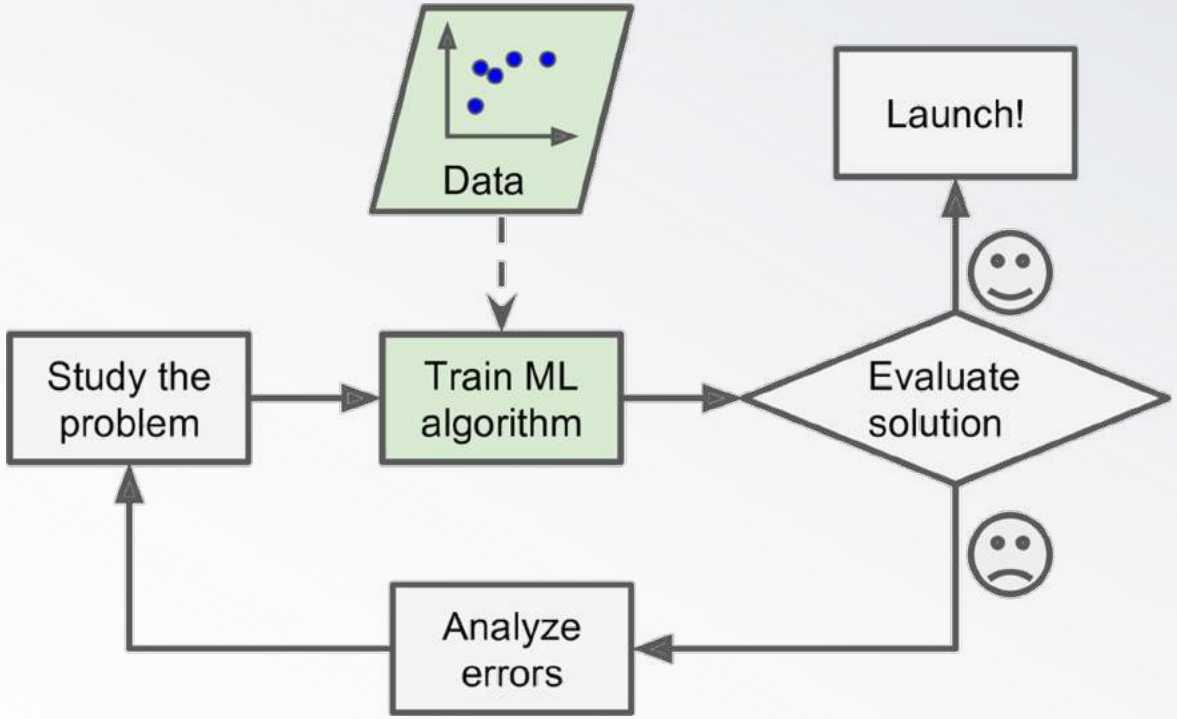


Deep Learning with Python
by François Chollet

Machine Learning

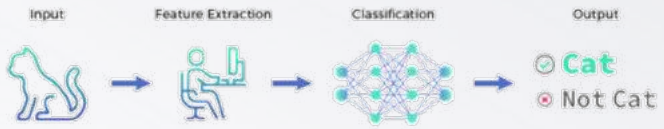
A new programming paradigm



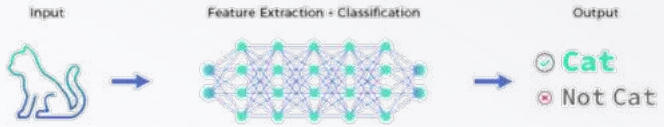


LEVERAGING DEEP LEARNING TO ACCOMPLISH NON-LINEAR TASKS

MACHINE LEARNING

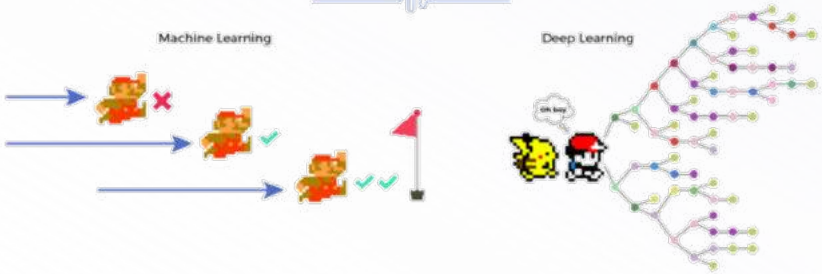


DEEP LEARNING



	Machine Learning	Deep Learning
Accuracy	Moderate with a false positive rate at up to 5%	Extremely high with false positive rate at nearly zero
Involvement of Domain Expert	Required for feature engineering & extraction	Not required
Data Set That Can Be Analyzed	Only 2.5-5% of available data	Processes 100% of available raw data
Correlations	Only simple linear correlations	Non-linear correlations, i.e. correlations that exist in complex patterns, rather than simple1-1 correlations

MACHINE LEARNING VS. DEEP LEARNING VISUALIZED



https://www.rocketsource.co/blog/machine-learning-models/

ROCKET SOURCE

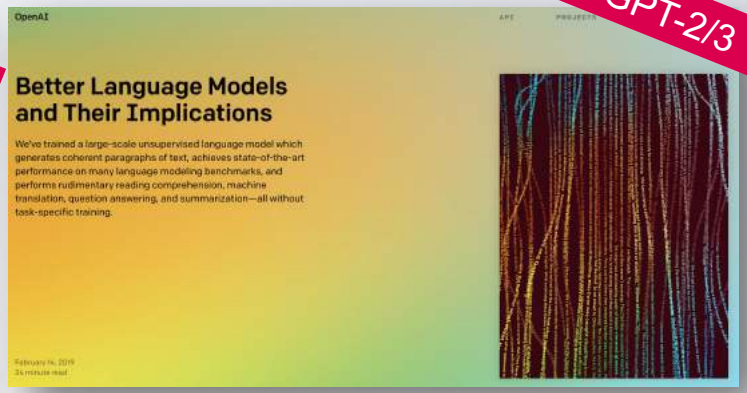
Machine Learning and Deep Learning

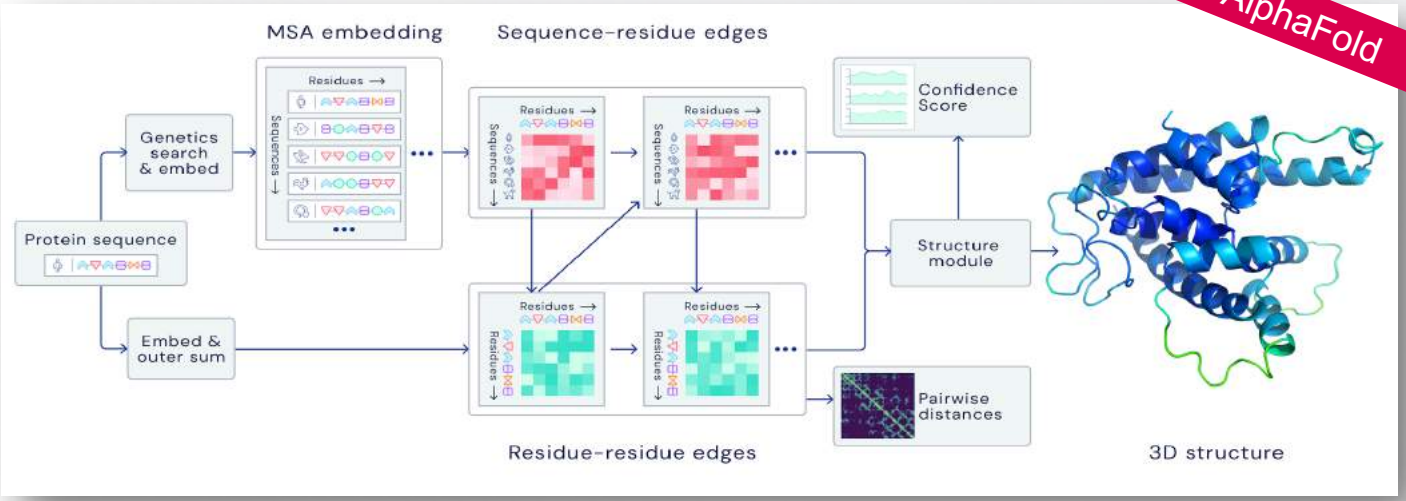
Künstliche Intelligenz ist wieder en vogue – Dank Deep Learning

Unter Künstliche Intelligenz fallen eigentlich ganz unterschiedliche Bereiche

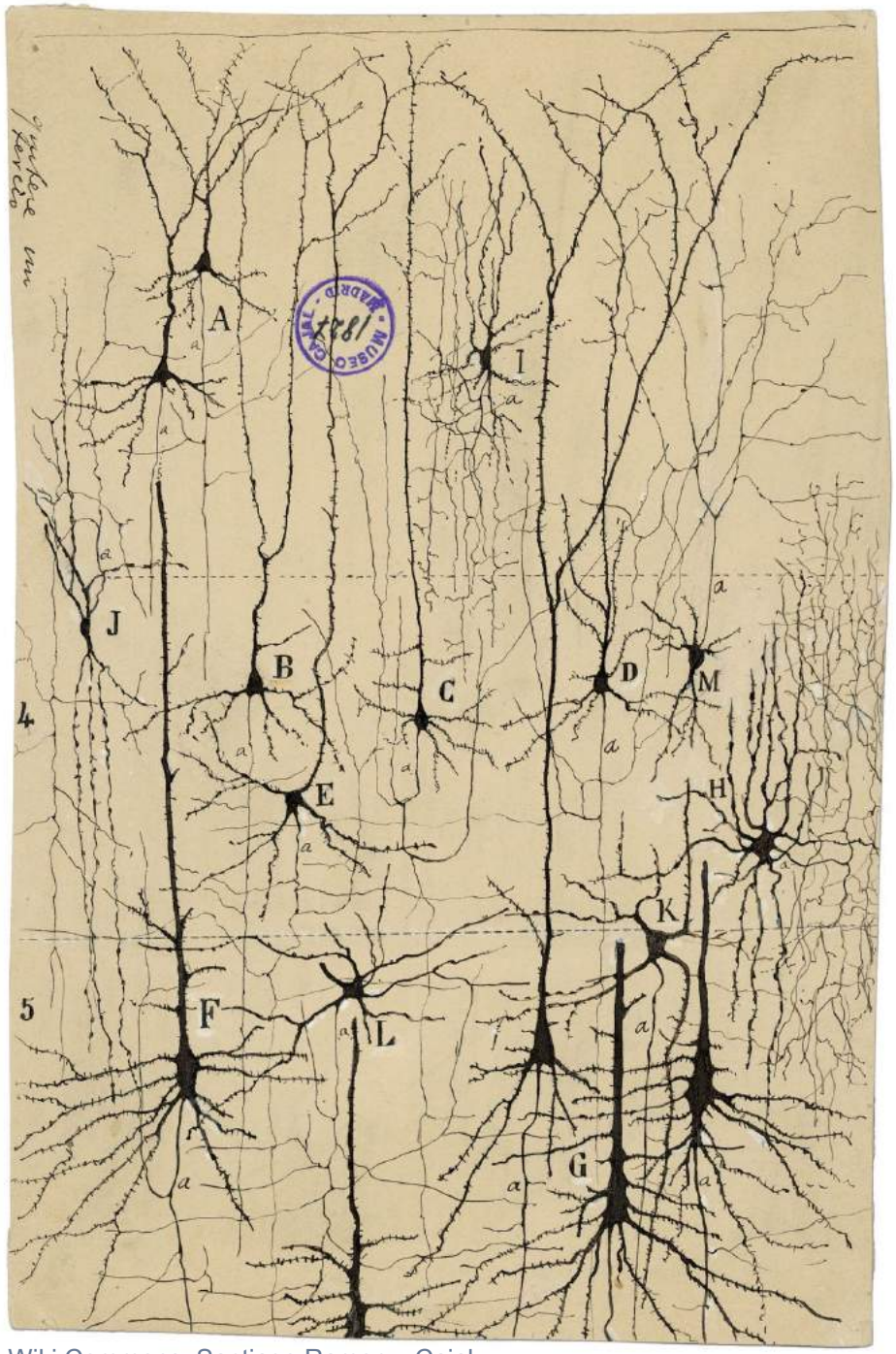
Quellen:
https://www.kansascity.com/news/business/technology/917xpi/picture62197987/ALTERNATES/FREE_640/atlas%20from%20boston%20dynamics
Wiki Commons: Computer vision sample in Simón Bolivar Avenue, Quito.jpg
https://towardsdatascience.com/a-brief-introduction-to-neural-style-transfer-d05d0403901d
Wiki Commons, Grafiker 61 – Own Work
Wiki Commons, Anon_lastname – Own Work

Spektakuläre Ergebnisse der letzten Jahre im Deep Learning





Quellen:
https://www.youtube.com/watch?v=WXuK6gekU1Y
https://openai.com/blog/tags/five/
https://openai.com/blog/better-language-models/
https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology
https://openai.com/blog/dall-e/
https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology



Wiki Commons: Santiago Ramon y Cajal

Artificial Neuron

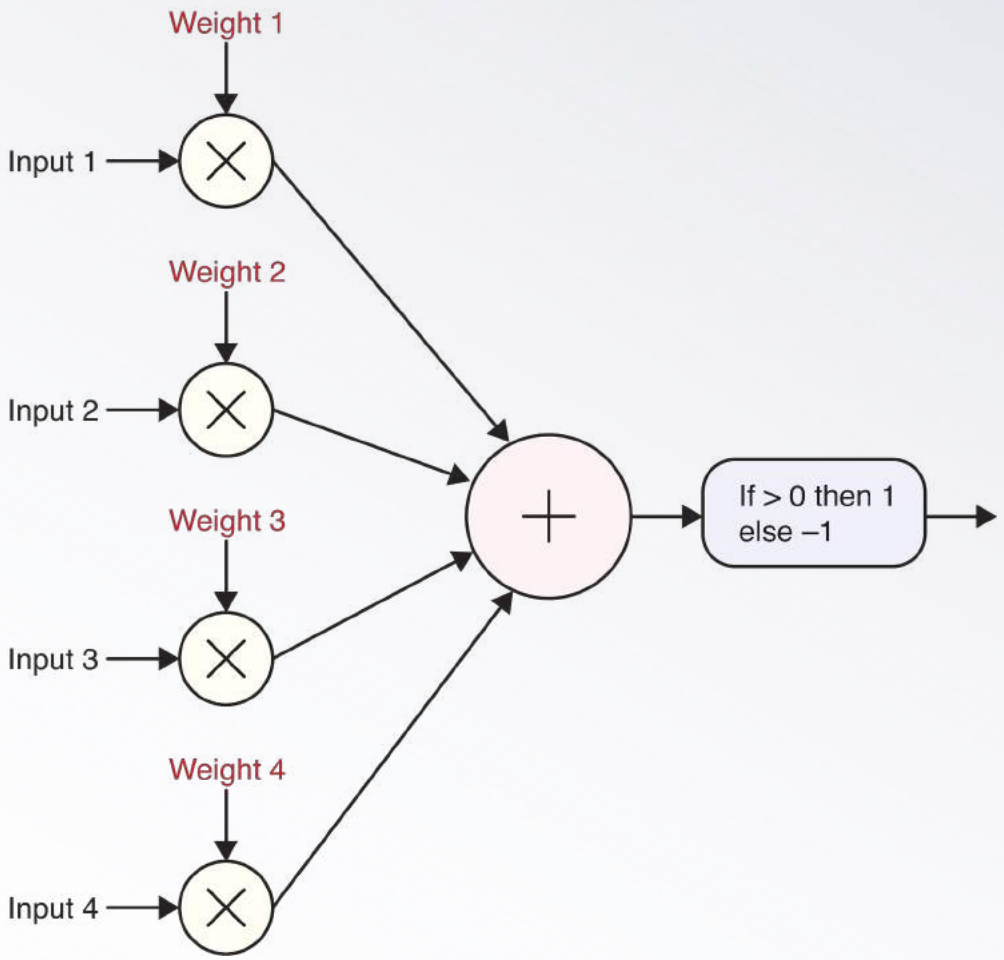
- 1943 Warren McCulloch und Walter Pitts
- Threshold Logic Unit oder Linear Threshold Unit
- Eine Version von „integrate and fire“

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

Rosenblatt's Perceptron

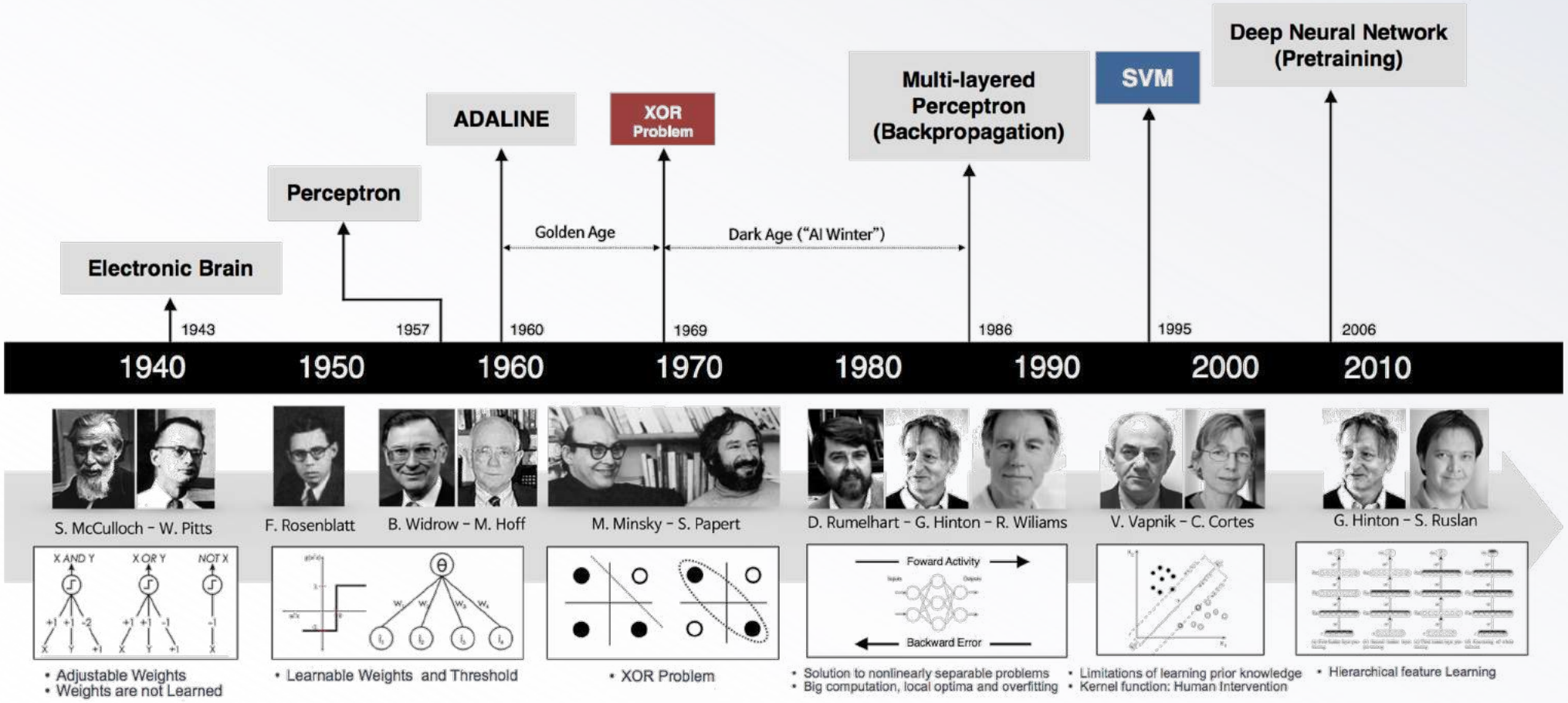
A simplified mathematical model of a neuron, 1962

1. Every input to a perceptron is represented by a single floating-point number
2. Each input is multiplied by a corresponding floating-point number called a **weight**
3. The results of these multiplications are all added ogether
4. Finally, the result is compared to a threshold value



Source: Deep Learning: A Visual Approach, Andrew S. Glassner, 2021, No Starch Press

Key events from the history of neural networks



Source: https://qingkaikong.blogspot.com/2016/11/machine-learning-3-artificial-neural.html, 2021-08-10

Backpropagation ‘86

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

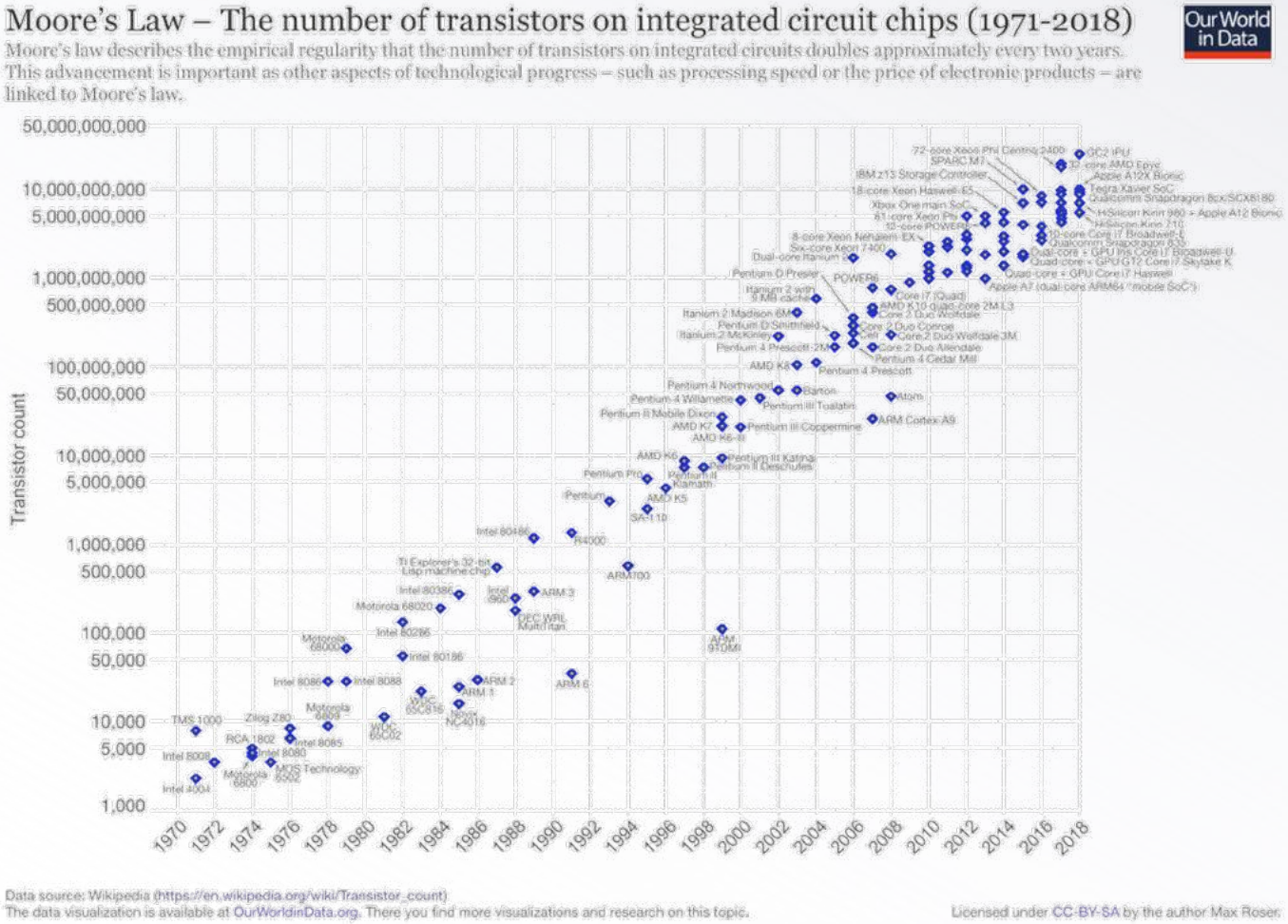
There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \qquad (1)$$

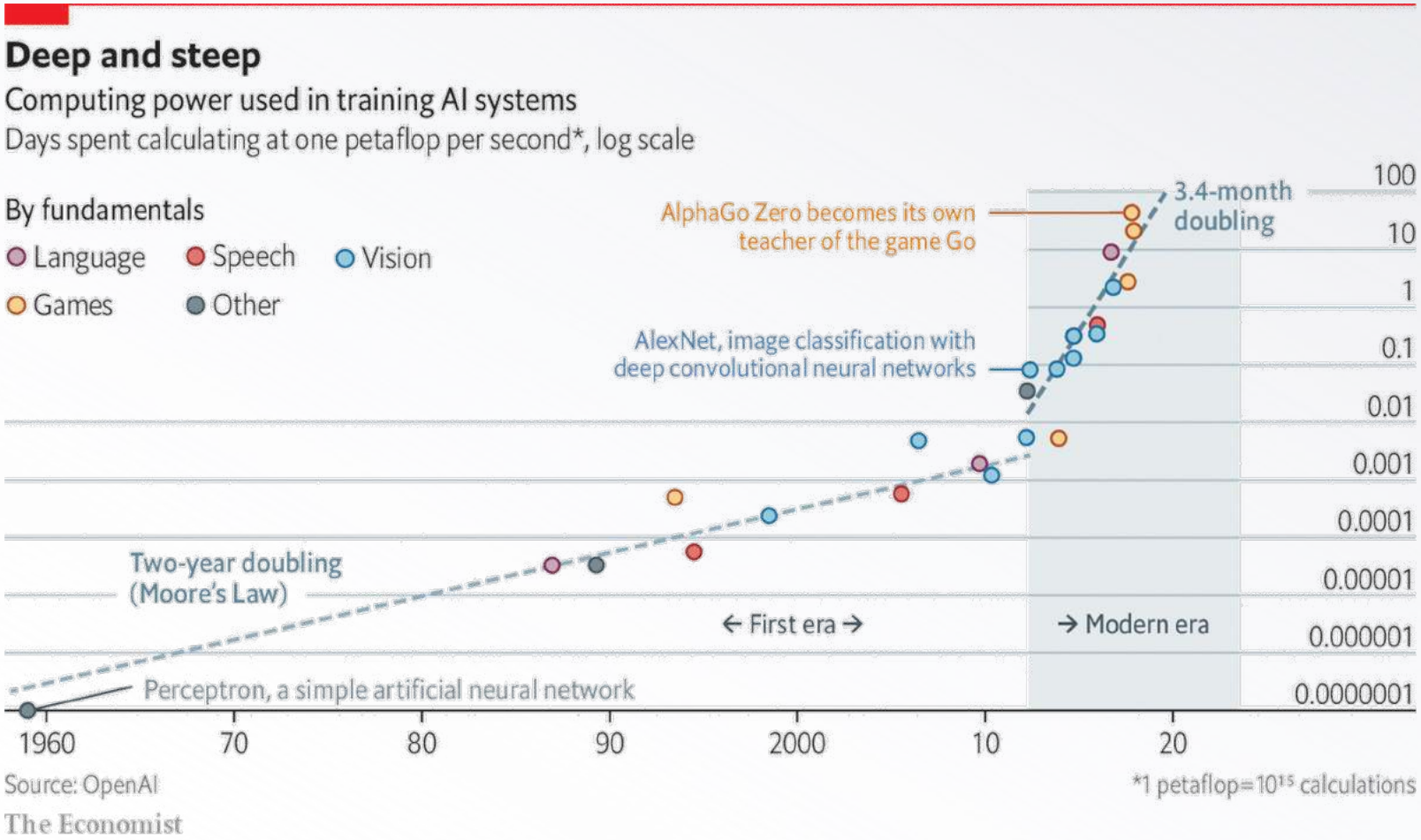
Moore‘s Law & GPUs



Source: wiki commons: Max Roser, https://ourworldindata.org/uploads/2019/05/Transistor-Count-over-time-to-2018.png

AI and Compute

Since 2012, the amount of compute has been increasing exponentially with a 3.4-month doubling time



Source: OpenAI (https://openai.com/blog/ai-and-compute/), Figure (http://www.jec.cl/articulos/?p=140249)

Deep Learning Durchbrüche - Bilderkennung

Source: www.valossa.com

Deep Learning Durchbrüche - Spracherkennung

Source: www.amazon.com

Deep Learning Durchbrüche – Reinforcement Learning

Source: www.alphagomovie.com

Deep Learning Durchbrüche – Reinforcement Learning

Source: www.openai.com

Beispiele für aktuelle Deep Learning Anwendungen

GENERATIVE ADVERSARIAL NETWORKS (GAN)

https://thisxdoesnotexist.com

HANDZEICHNUNGEN ERKENNEN MIT NEURONALEN NETZEN

https://quickdraw.withgoogle.com

TEACHABLE MACHINE IM BROWSER

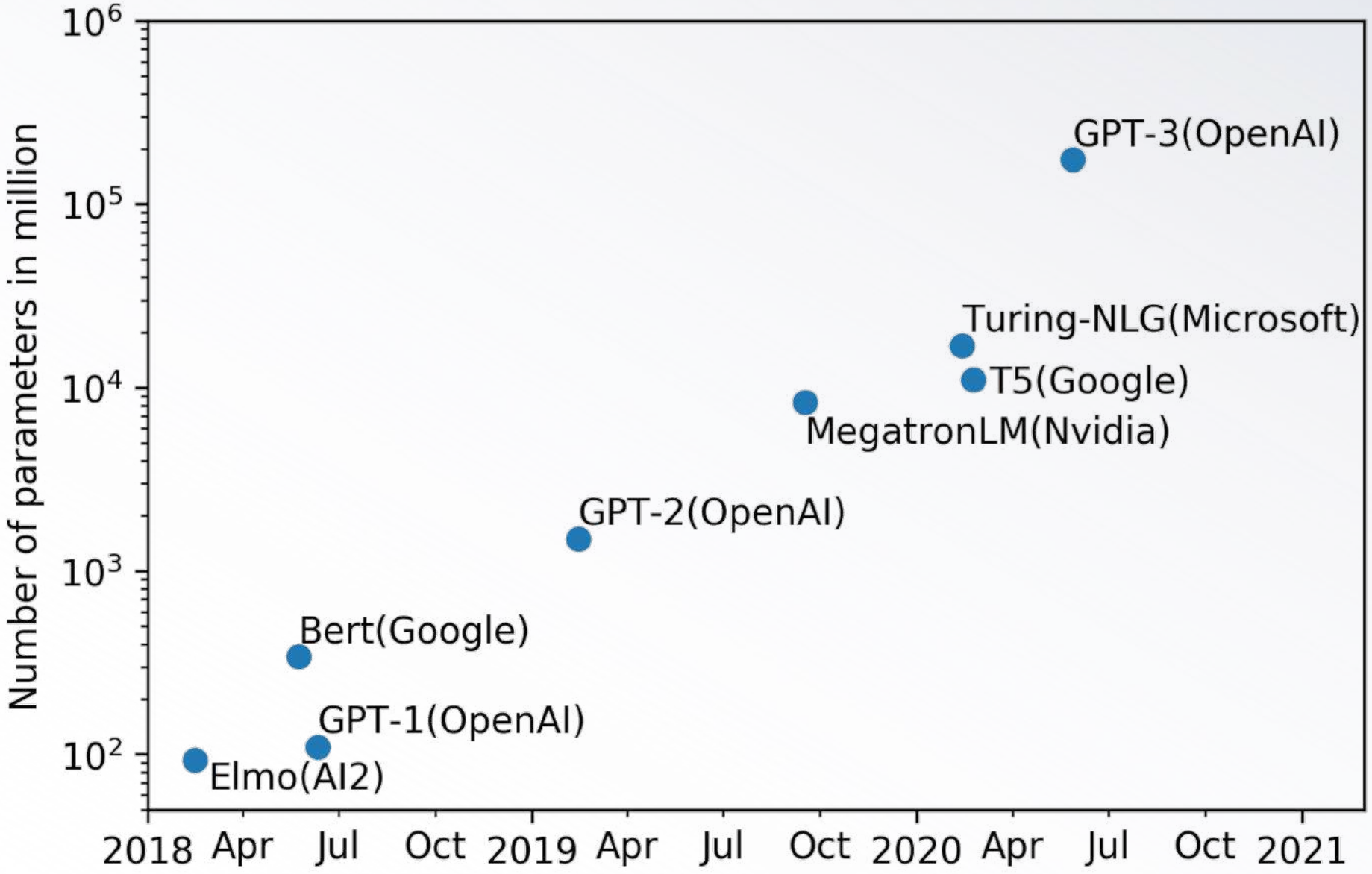
https://teachablemachine.withgoogle.com/

Vier Trends in der Deep Learning Forschung

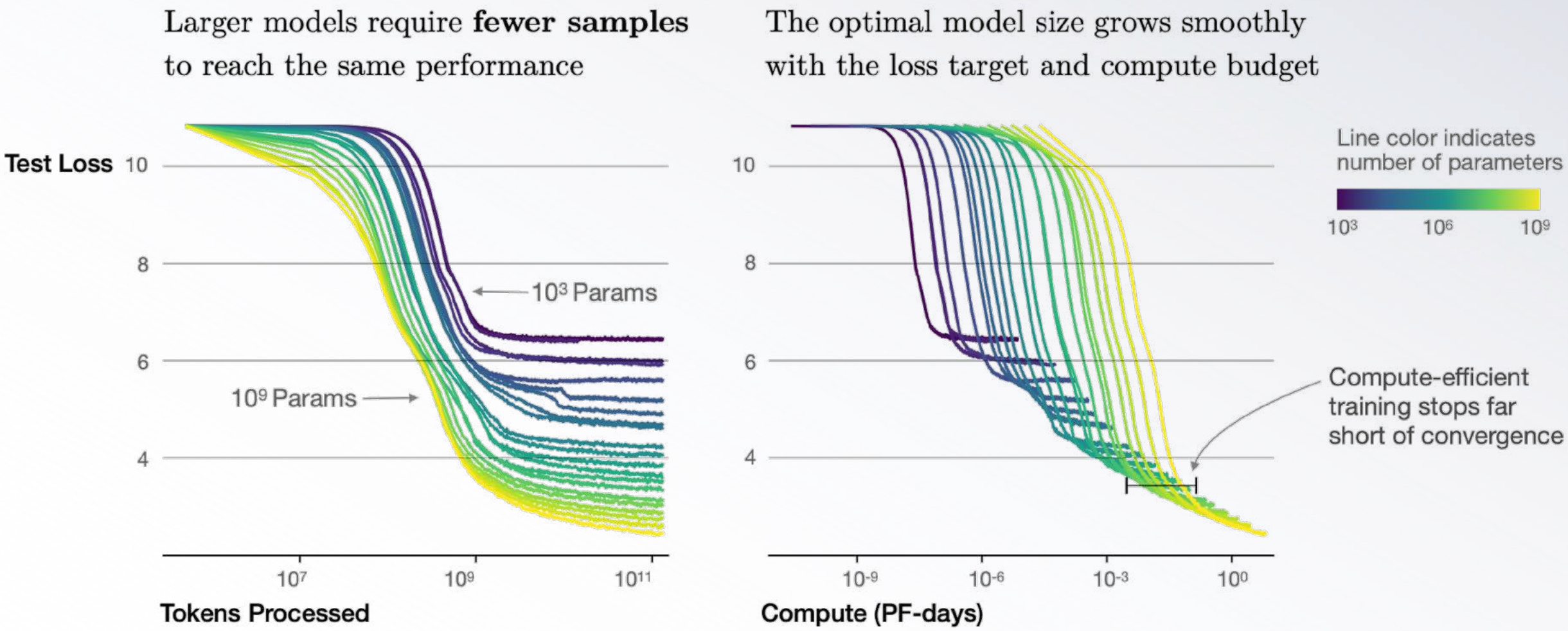
- Skalierung zu extrem großen Netzwerken
- Unsupervised Learning statt Supervised Learning
- Multimodales Training
  https://openai.com/blog/multimodal-neurons
- Generative Modelle
  https://thisxdoesnotexist.com

OpenAI

Seit der Einführung der Transformer-Architektur ist die Größe der Modelle exponentiell gestiegen



Die Qualität des Outputs verbessert sich vorhersehbar mit Parameteranzahl, Compute und Daten.



Source: Scaling Laws for Neural Language Models, 2020, arXiv:2001.08361

PRAXIS

Einstieg in DL – Erstkontakt mit MNIST

2

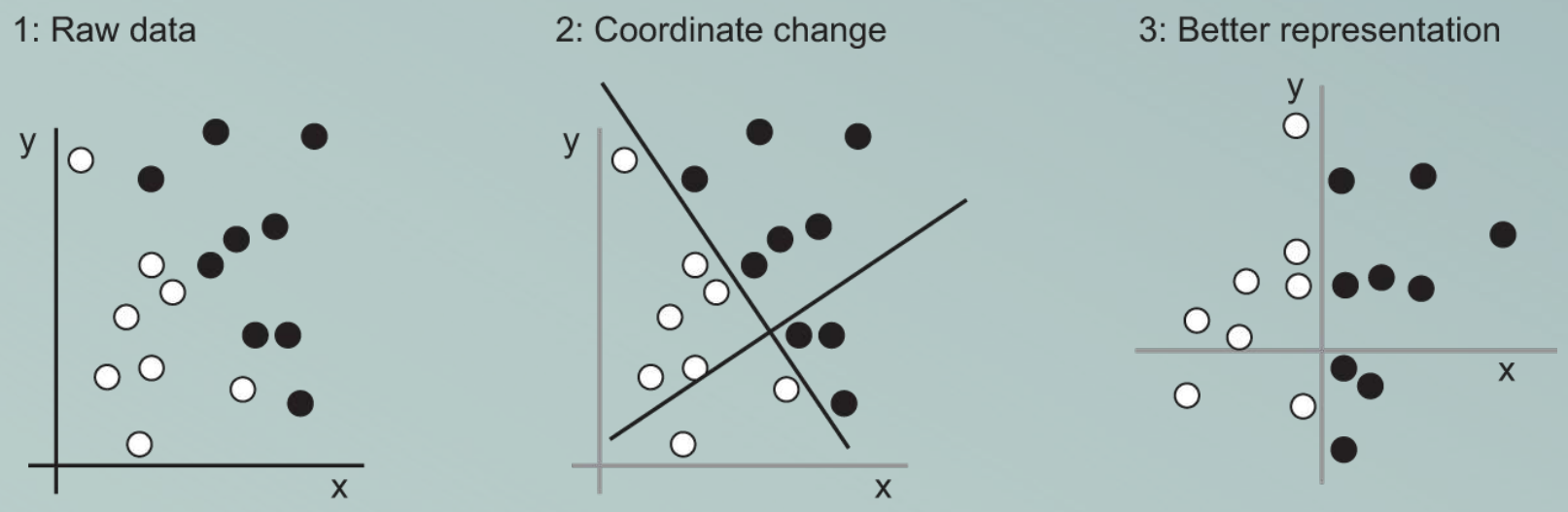
The world of tensors

Data representation in deep learning

What is a Data Representation?

It's a different way to look at data – to represent or encode data.

Machine learning models are all about finding appropriate representations for their input data---transformations of the data that make it more amenable to the task at hand.



Was ist ein Tensor?

- Tensoren sind Verallgemeinerungen von Matrizen und Vektoren.
- Ein Skalar -1 ein Rang-0 Tensor
- Ein Array [0, -1, -2, -3] ist ein Rang-1 Tensor
- Eine Matrix ist ein Rang-2 Tensor

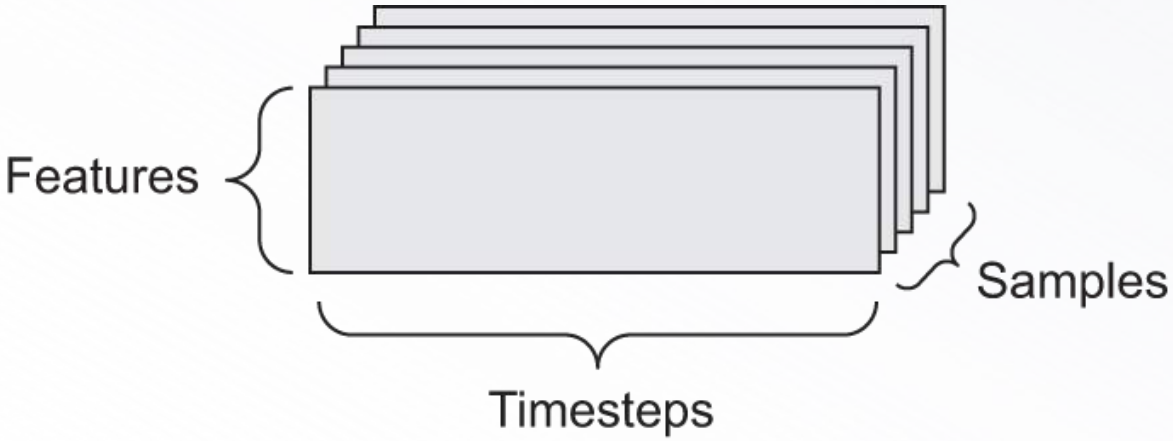
$$\mathbf{A} = \begin{bmatrix} 0 & -1 & -2 & -3 \\ 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 \end{bmatrix}$$

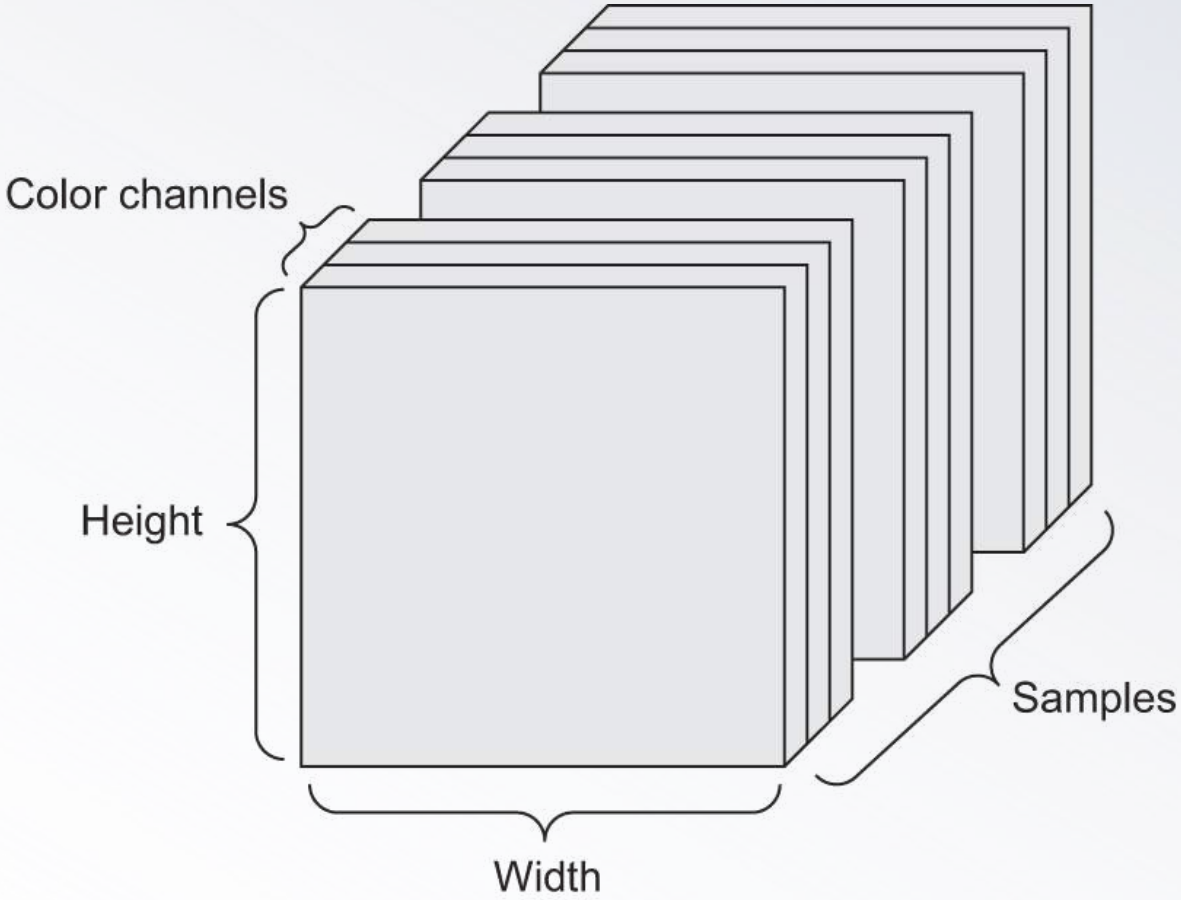
- Vorsicht: Mathematischer Rang einer Matrix != Tensor-Rang
- Vorsicht2: Numpy-Dimension aka x.ndim = Rang
- Rang wird auch Anzahl der „Axes“ genannt

Tensor-Beispiele verschiedener Ränge

- Rang-0: Skalare wie z.B. Temperatur
- Rang-1: Arrays oder Vektoren, wie z.B. eine Liste von Eigenschaften
- Rang-2: Matrizen, wie z.b. Time Series von Vektoren (Zeit axis = zweite)
- Rang-3: Ein Farbbild (Höhe, Breite, Farbkanal) oder Grautonbilddatensatz
- Rang-4: Ein Farbbilddatensatz, typisch: (Sample, Farbkanal, Höhe, Breite)
- Rang-5: Videodaten (sample, Zeit, Höhe, Breite, Farbkanal)

Tensor-Beispiele verschiedener Ränge





Tensor Eigenschaften

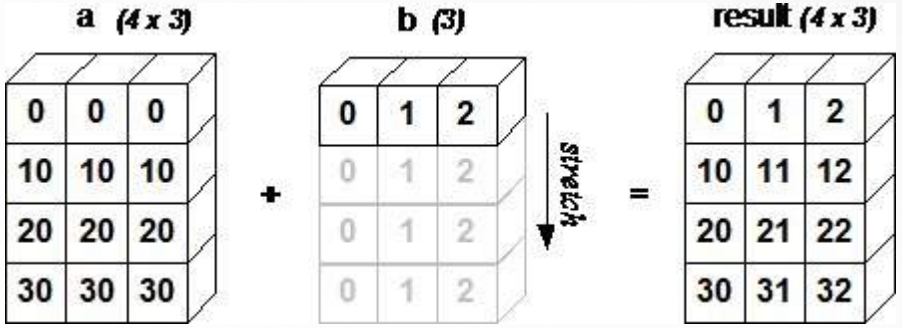
- Rang bzw. Anzahl der “Axes“
- Erste „Axis“ = Samples (aka Datenpunkte oder Beispiele)
- Shape, d.h. wie viele Dimensionen in jeder “Axis“
- Datentype, dtype = float16, float32, uint8, etc.
- Tensor X:
  - X.ndim
  - X.shape
  - X.dtype
- Slicing – Teilmengen der Daten auswählen
  - teilmenge = train_images[:1000,:,:]
  - batches

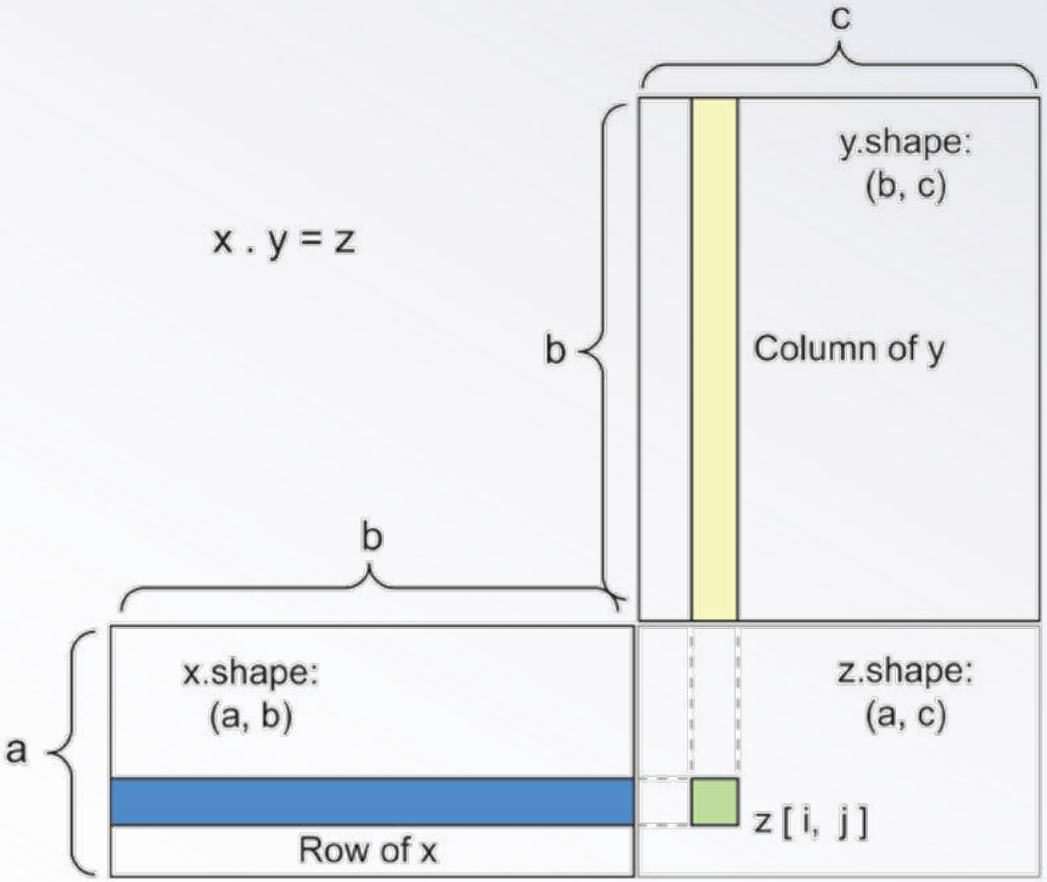
Tensor Operationen

- +,-,/,* und auch Funktionen werden typischerweise „elementwise“ interpretiert, d.h. die Operation wird auf jeden Eintrag einzeln angewandt.
- Stark optimiert, da sehr parallelisierbar
- dot product (aka Skalarprodukt, inneres Produkt)
- Broadcasting für Tensoren mit unterschiedlicher Shape
  - „Axes“ werden hinzugefügt bis der Rang gleich ist
  - Entlang der hinzugefügten “Axes“ wird der Tensor wiederholt

Broadcasting Beispiel

```
(a, b, c, d) • (d,) -> (a, b, c)
(a, b, c, d) • (d, e) -> (a, b, c, e)
```





Elementweises Broadcasting:

- Arrays haben dieselbe Form.
- Tensoren haben denselben Rang und die Länge jeder Achse ist entweder gleich oder 1.
- Tensoren mit zu niedrigem Rang, können Achsen mit Länge 1 vorne angefügt bekommen, bis Punkt 2 gilt.

PRAXIS

Datenrepräsentationen

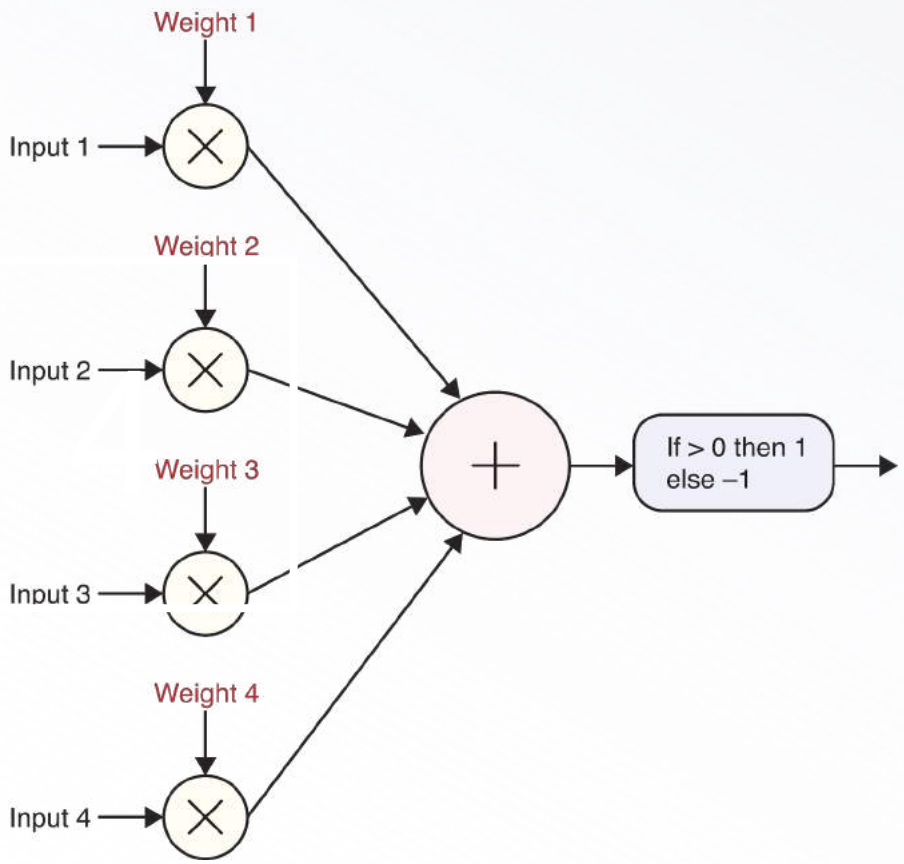
Tensoroperationen

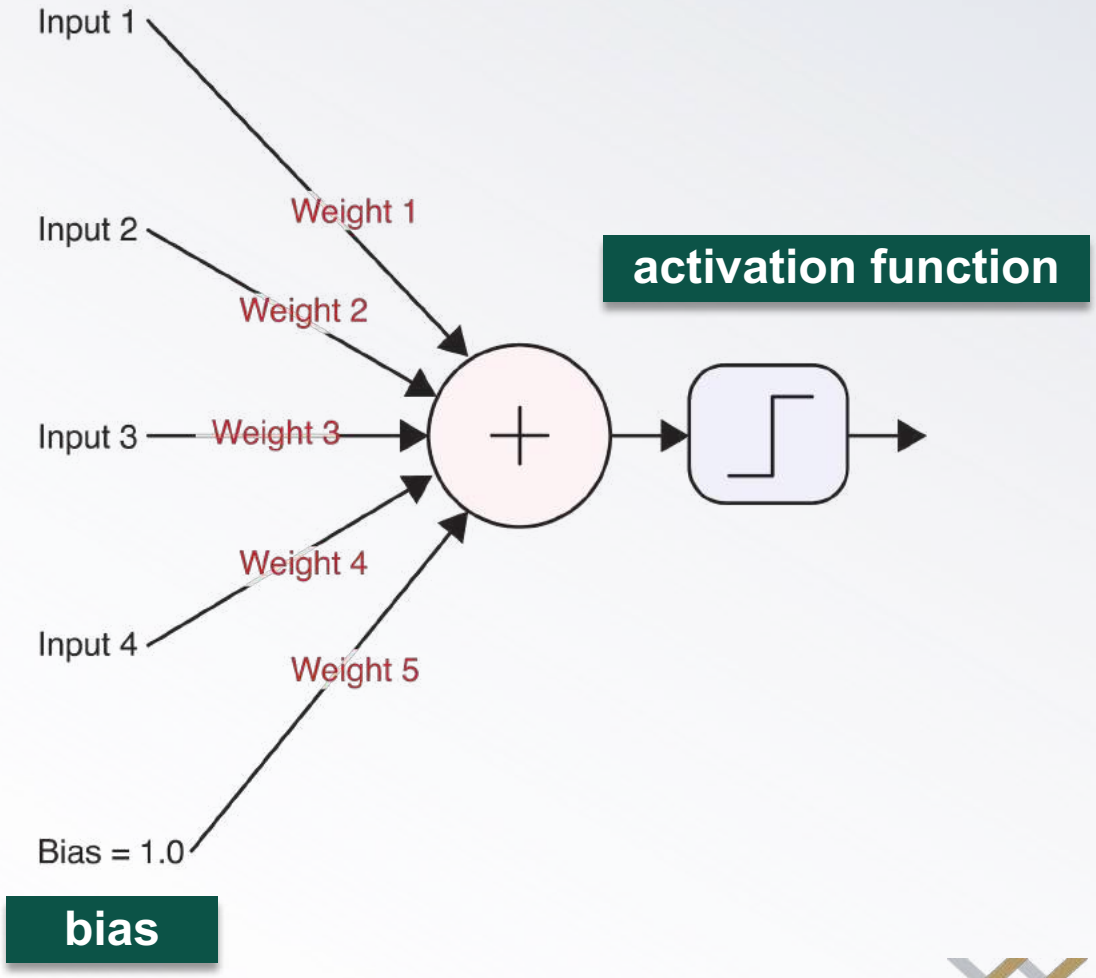
3

Uncrumple a paper ball

Geometric interpretation of deep neural networks

From the Perceptron to a modern neural network

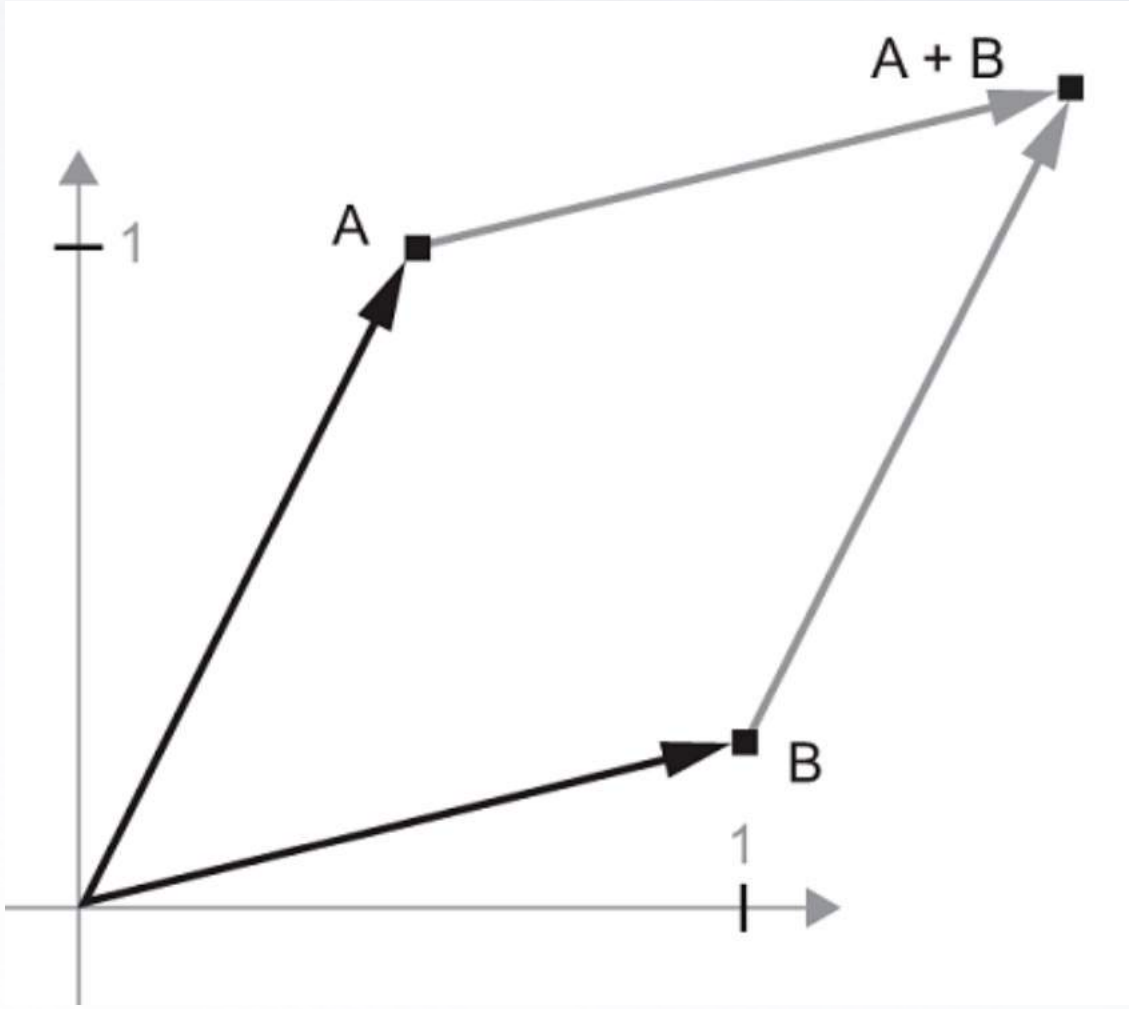




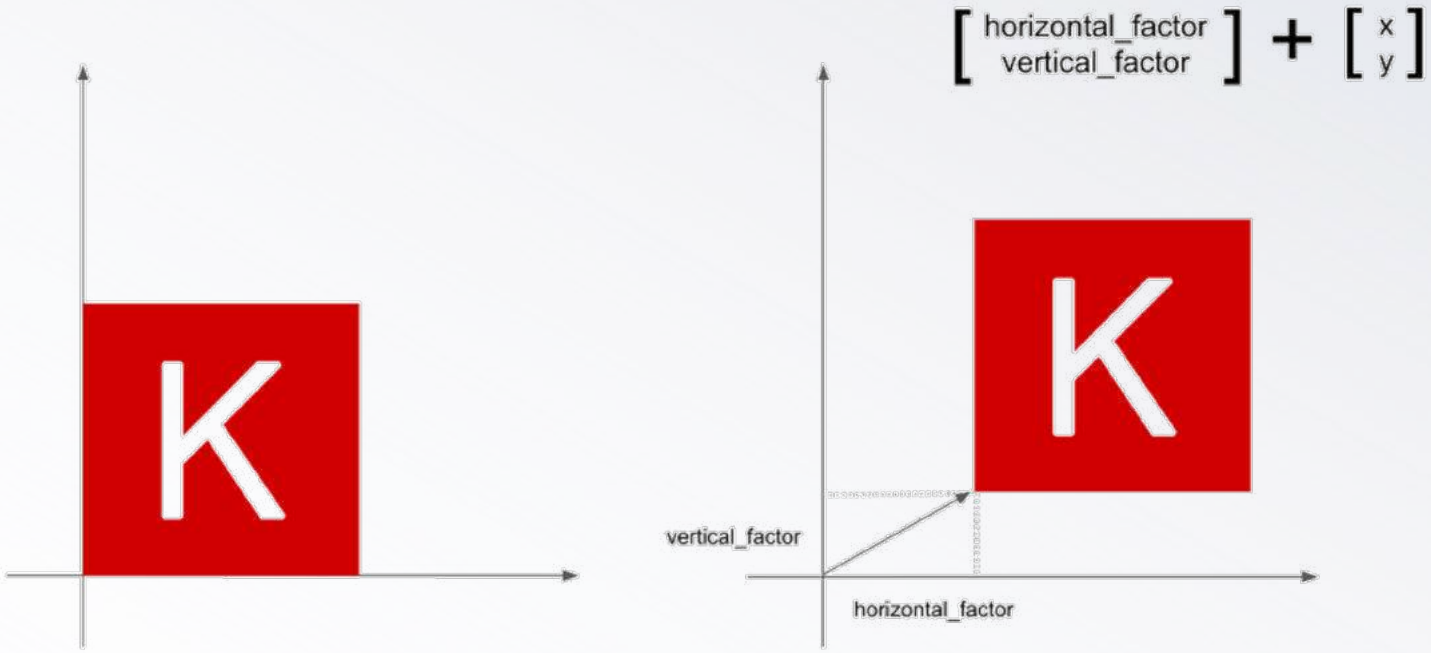
Geometrische Interpretation von Deep Learning - Affine Transformationen

Vektorarithmetik hat immer eine geometrische Interpretation:

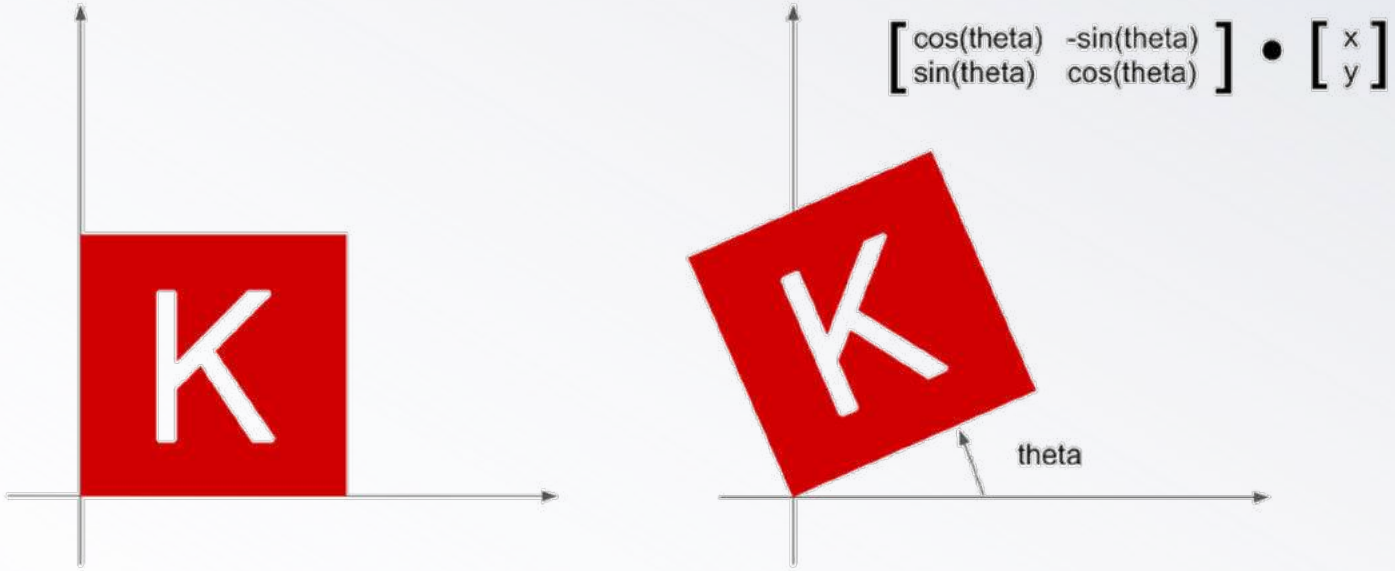
z. B. die Addition von Vektoren.



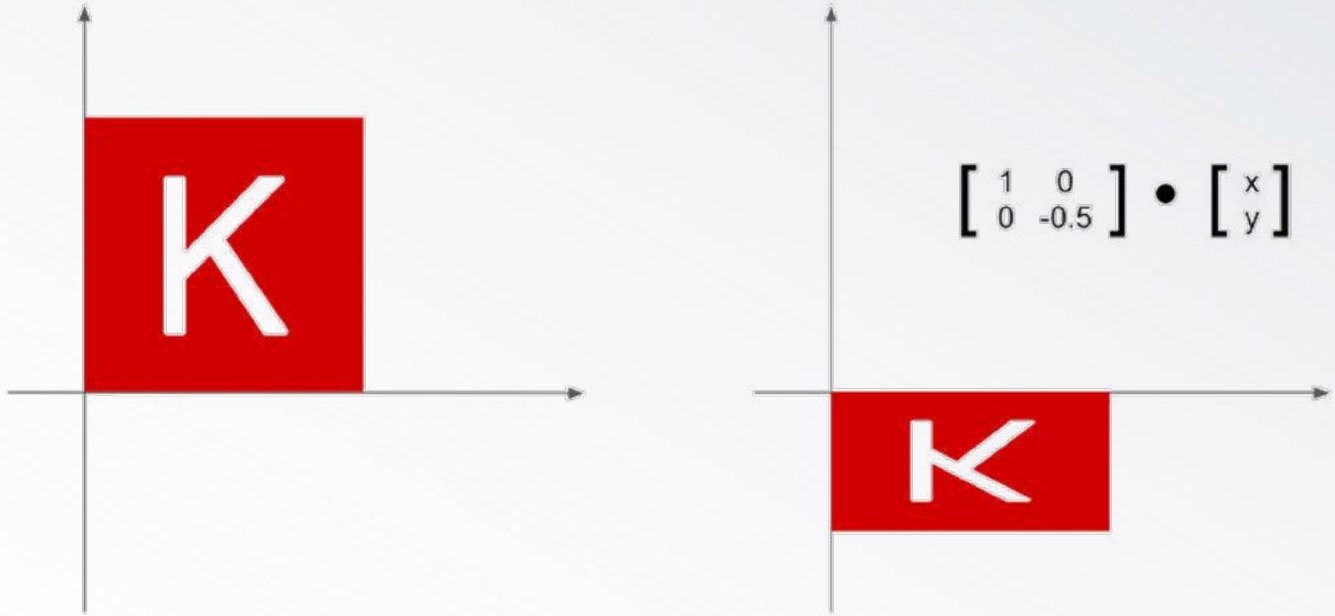
Einen Vektor zu einer Punktmenge zu addieren, nennt man **Translation**.



Durch eine Matrixmultiplikation kann man Punktmengen rotieren.

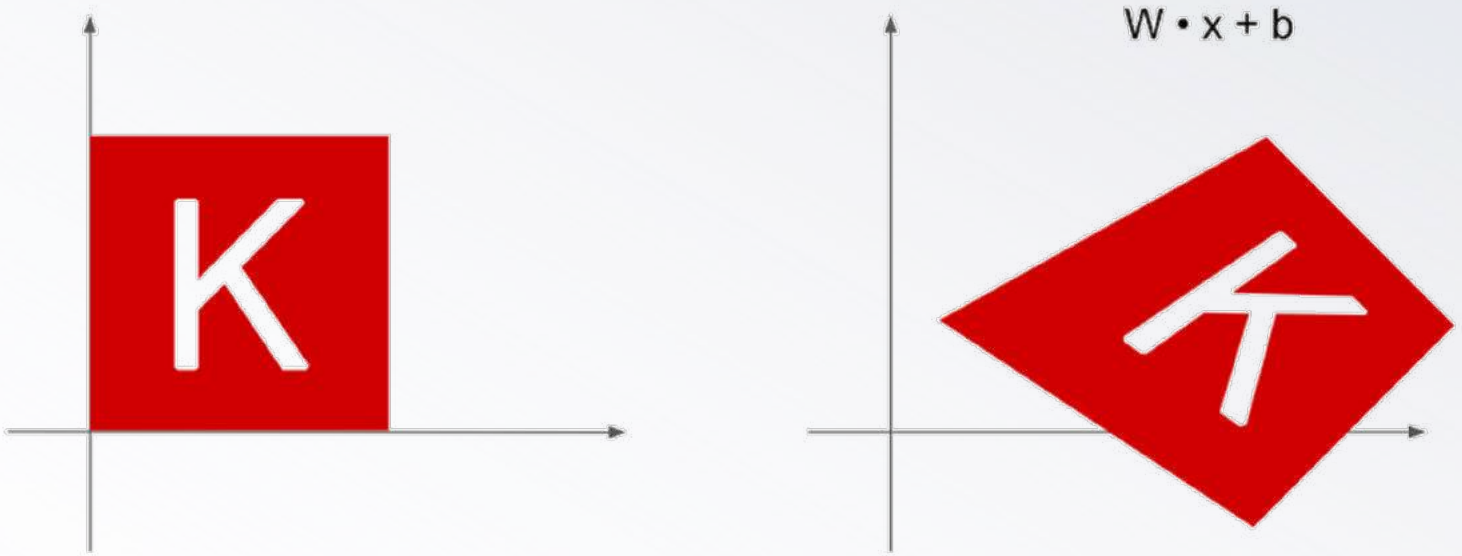


Durch eine Matrixmultiplikation kann man Punktmengen spiegeln und skalieren.



Wenn man diese Operationen verbindet, erhält man eine affine Tranformation.

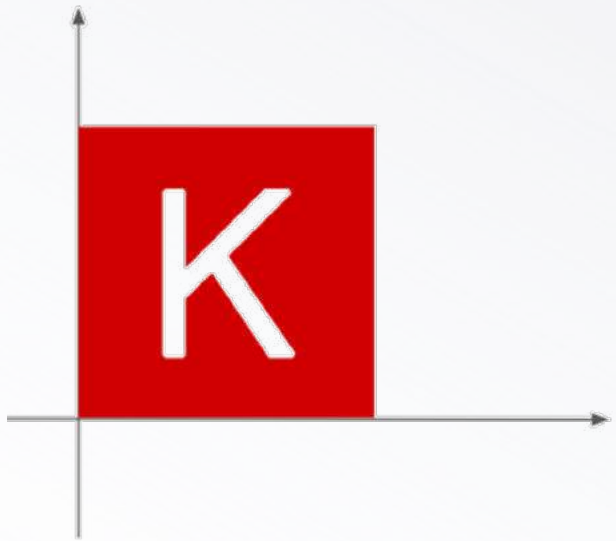
Affine Transformation = Lineare Transformation + Translation

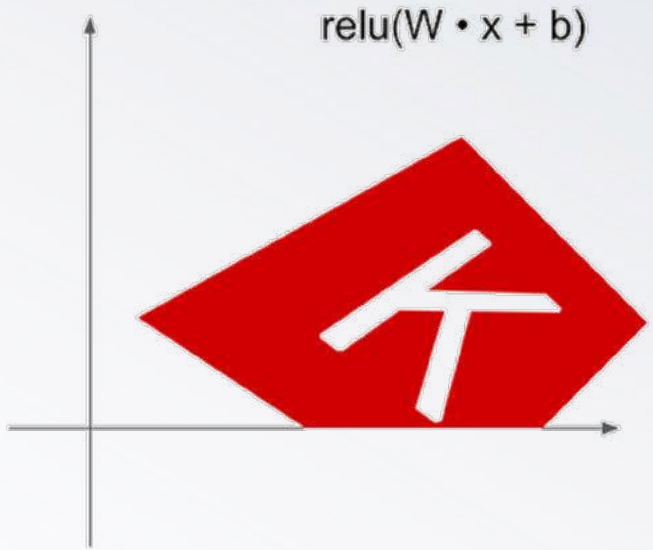


Der Input wird rotiert, gespiegelt, verzerrt und verschoben

Durch die Activation-Function wird die Transformation nicht-linear.

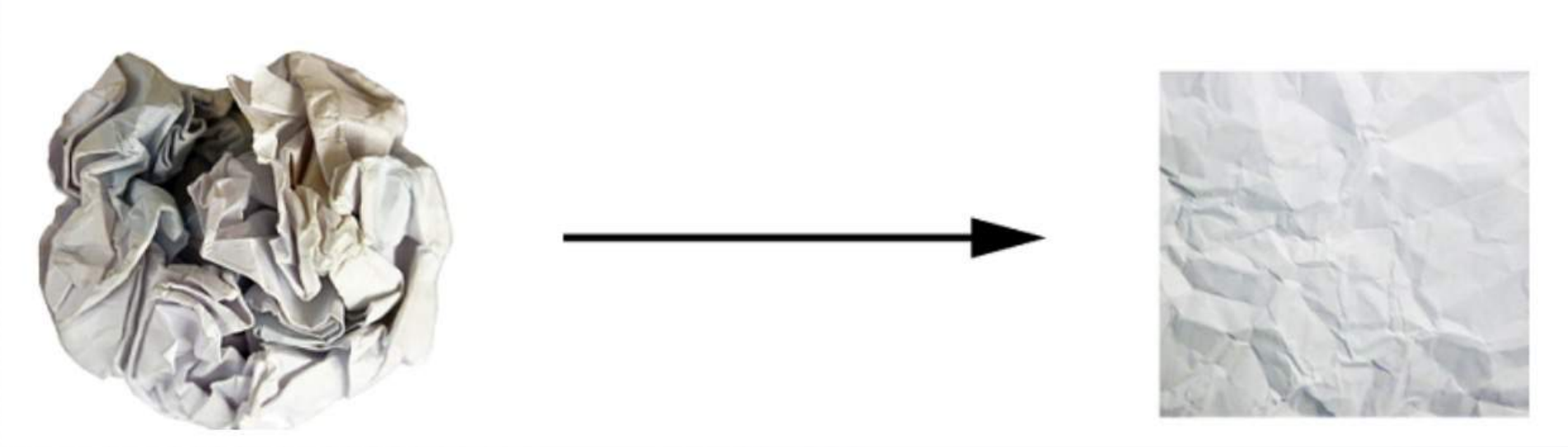
Aufbau eines Dense-Layers mit Aktivierungsfunktion.





Geometrische Interpretation von Deep Learning - Affine Transformation

Chollet vergleicht die Fähigkeit von Neuronalen Netzwerken mit dem Entknüllen zweier zerknüllter Papierblätter. Erst dadurch, dass sie Schritt für Schritt wieder auseinander gefaltet werden, lassen sie sich am Ende einfach trennen.

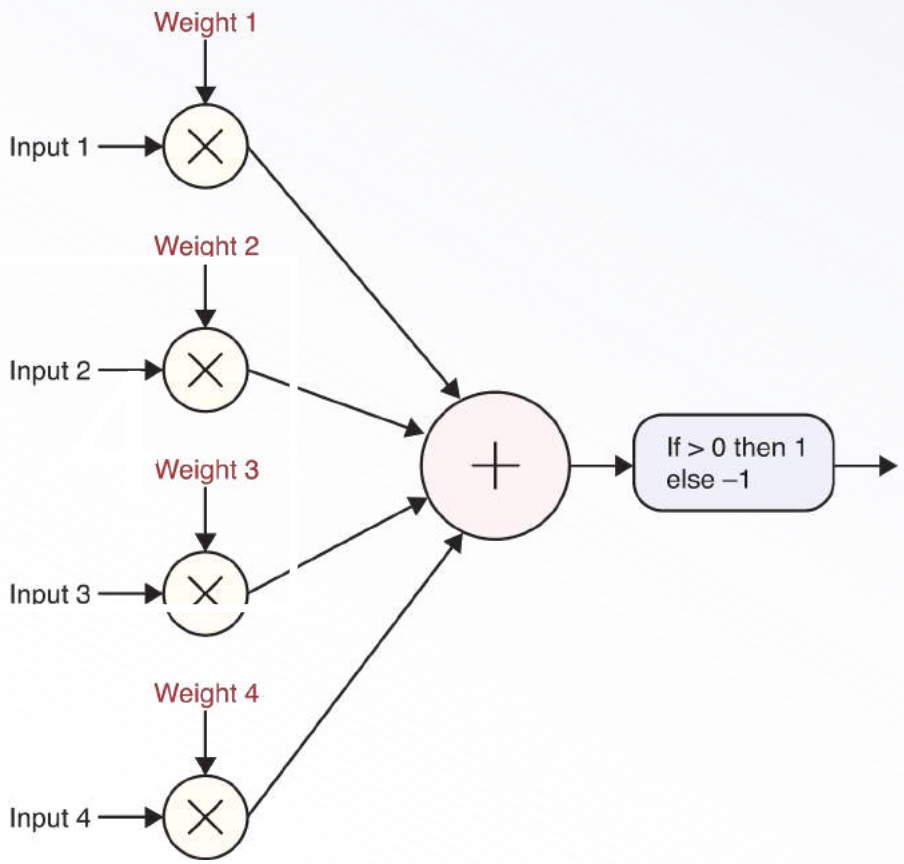


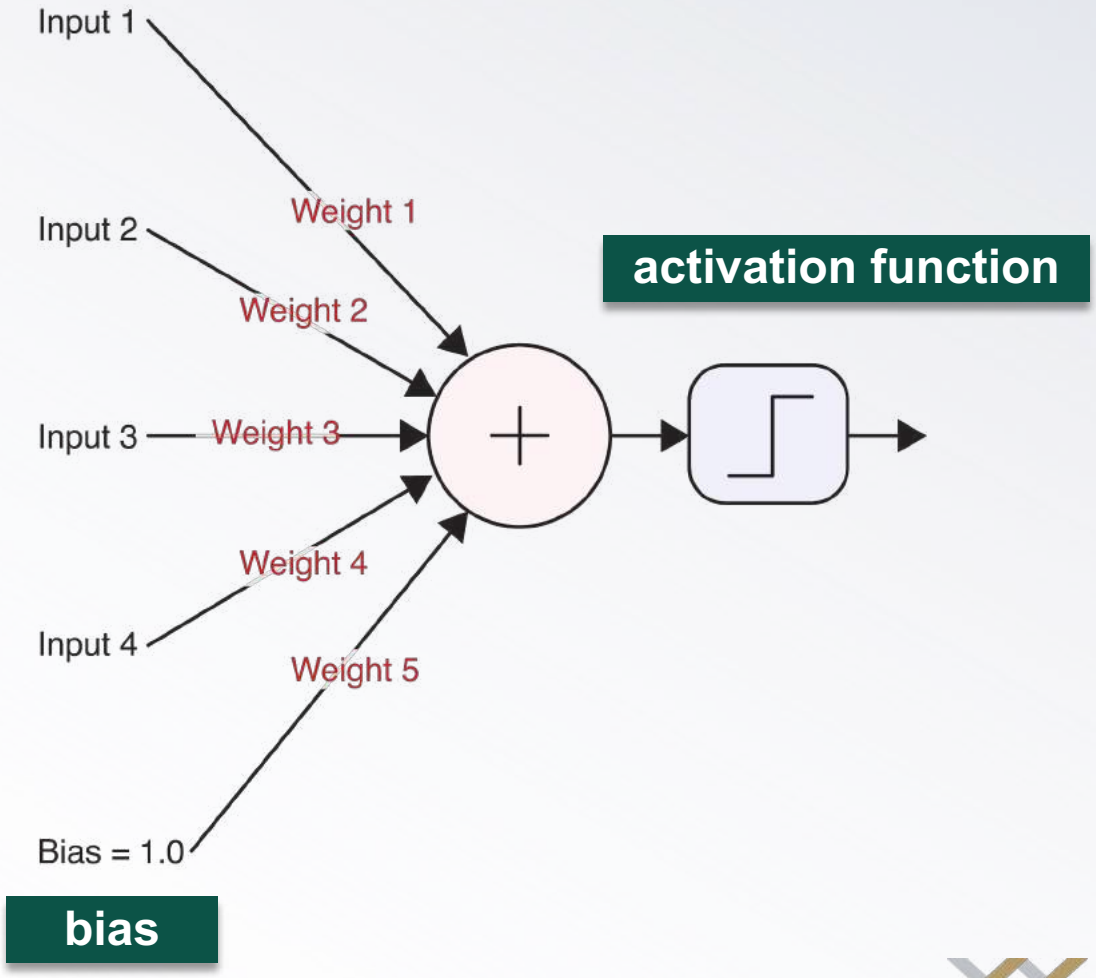
4

Understanding neural networks

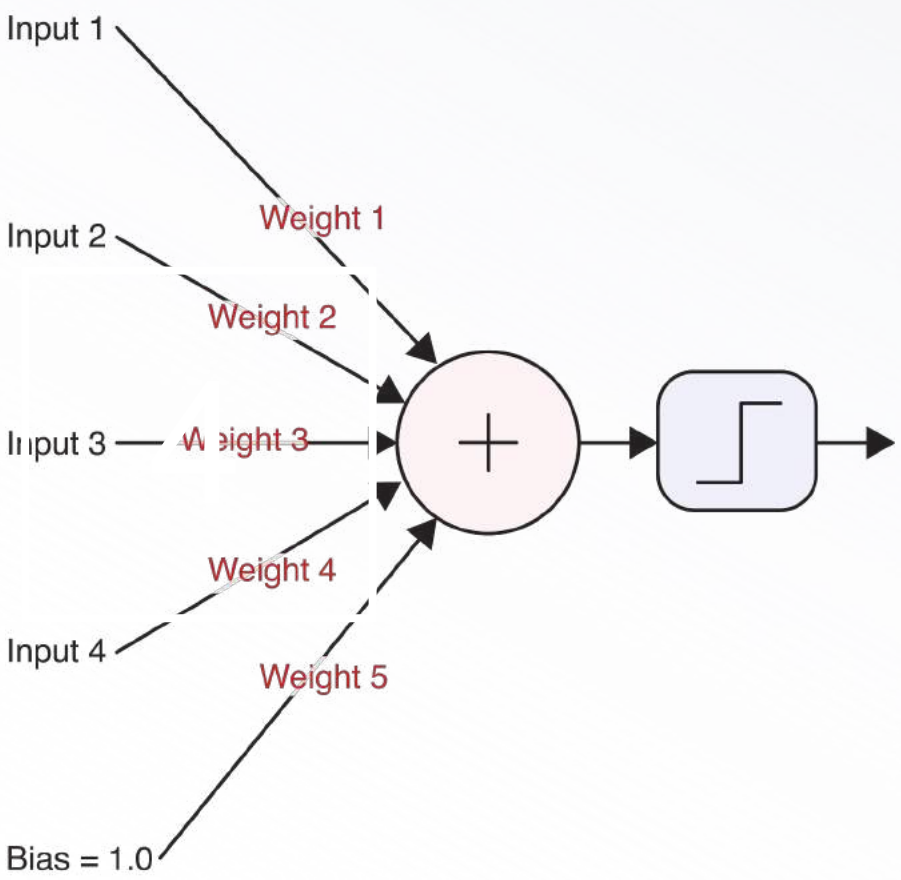
The building blocks in deep learning

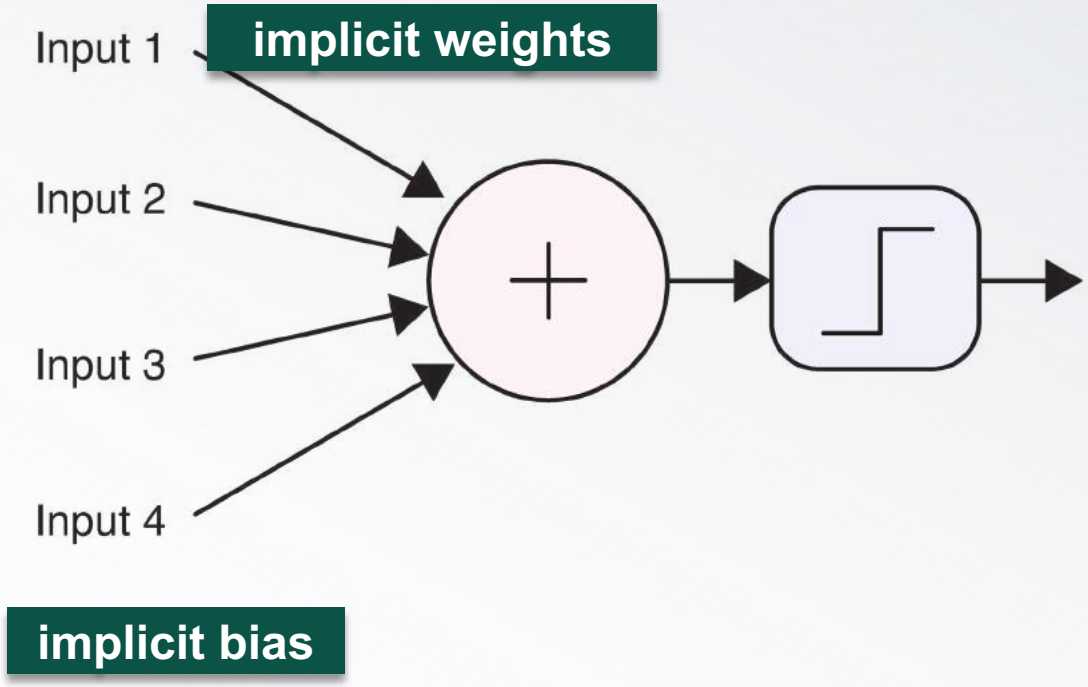
From the Perceptron to a modern neural network





From the Perceptron to a modern neural network

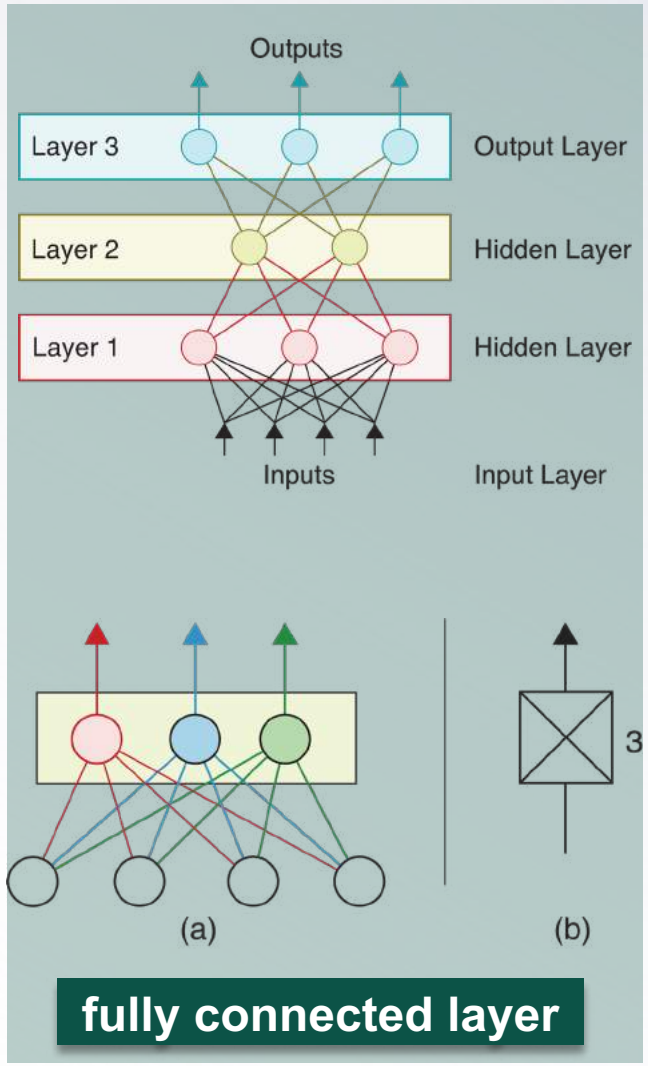


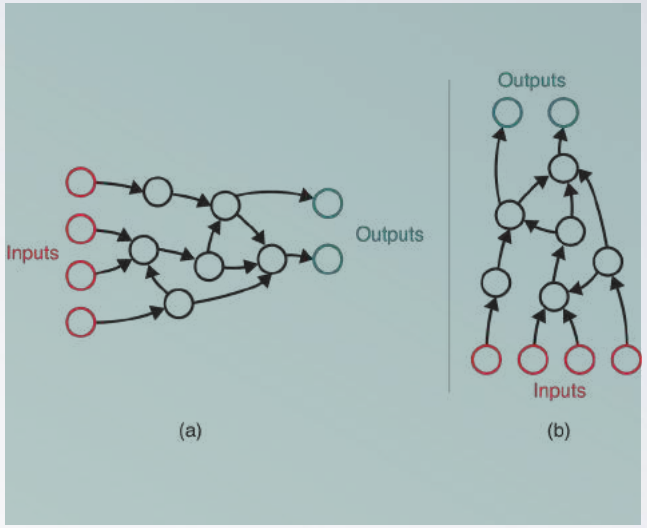


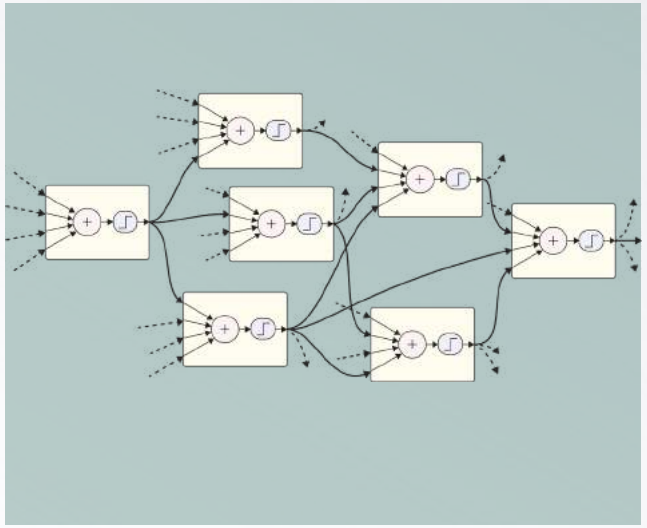
Hierarchical processing

Deep Networks

Deep networks arrange neurons into **layers**. Typically, neurons from one layer get their inputs only from the previous layer and send their output only to the following layer. Each layer builds on the work done by the previous stage.







Deep neural network architecture – Layers

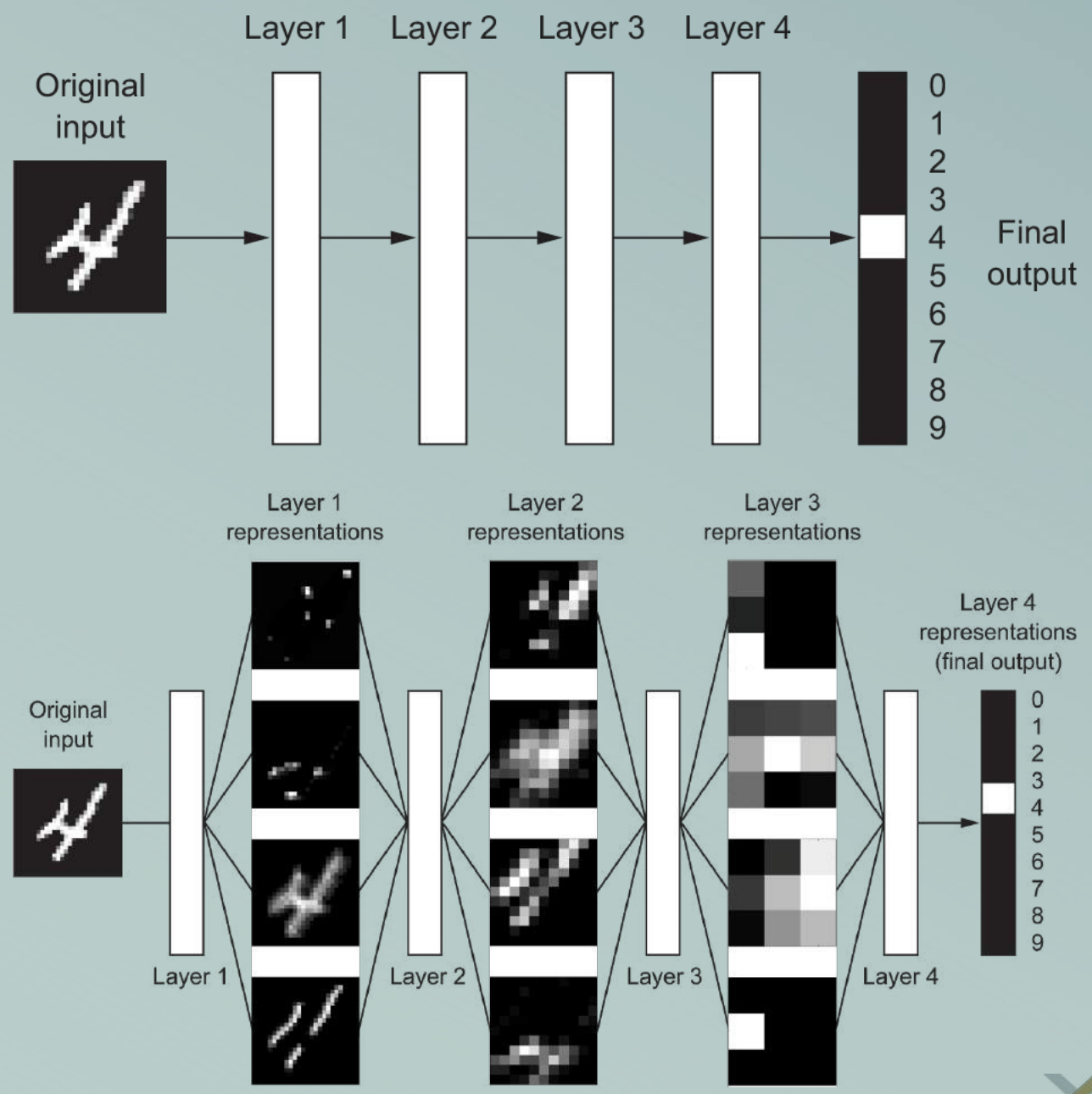
Layer	Intuition	Inductive bias	When to use
Fully connected	Allow all possible connections	None	Data without structure (i.e. tabular)
2D convolution	Recognizing spatial patterns	Local, spatial patterns	Spatial structure (i.e. images)
LSTM	Database	Sequences, memory	Data with sequential structure
Attention	Focus on similarities	Similarity, limit information flow	Data with sequential structure

Source: https://adgefficiency.com/guide-deep-learning/

“Deep” in Deep Learning

Deep Learning is a subfield of machine learning that puts an emphasis on learning successive layers of increasingly meaningful representations.

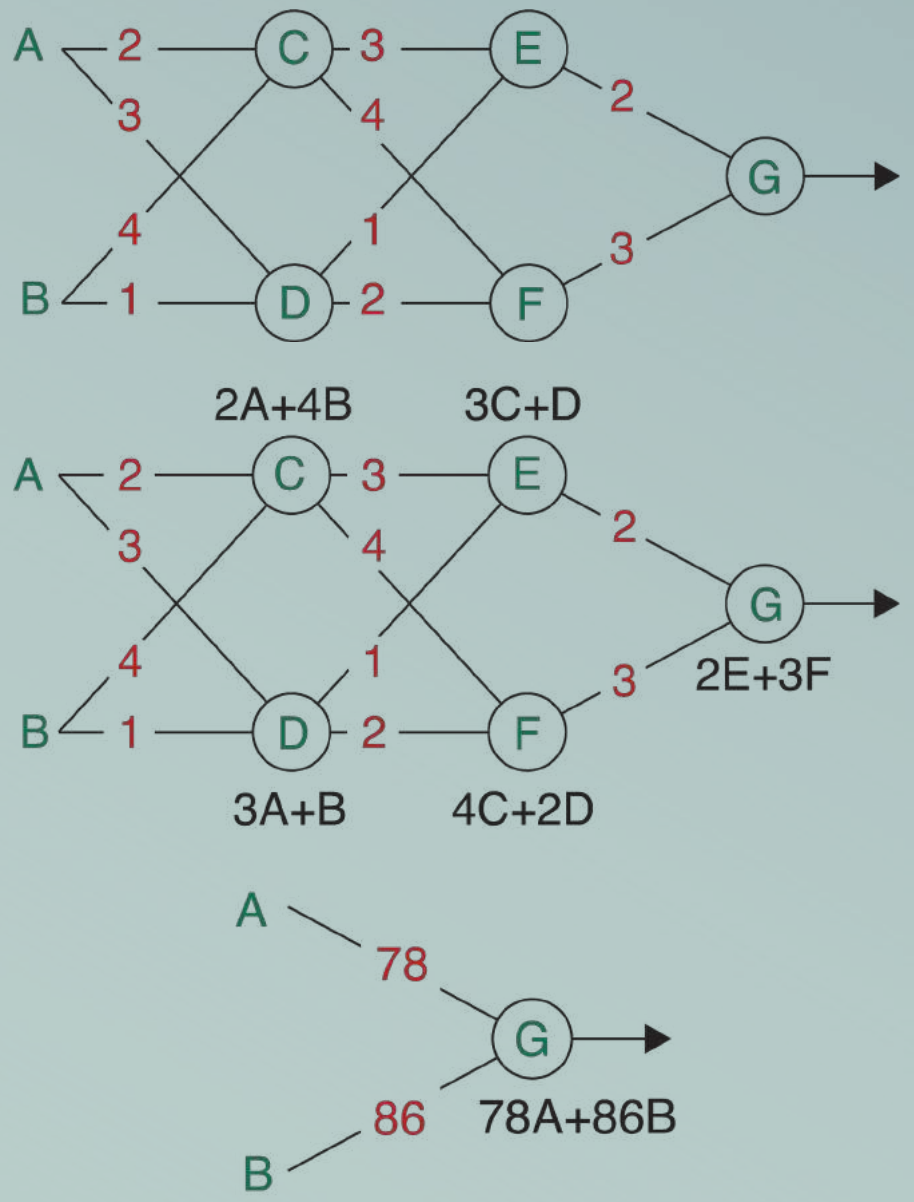
You can think of a deep network as a **multistage information-distillation process**, where information goes through successive filters and comes out increasingly purified.



Preventing Network Collapse

Without **activation functions**, the neurons in a network combine, or collapse, into the equivalent of a single neuron.

By designing activation functions to use **nonlinear** operations, we prevent this kind of collapse.

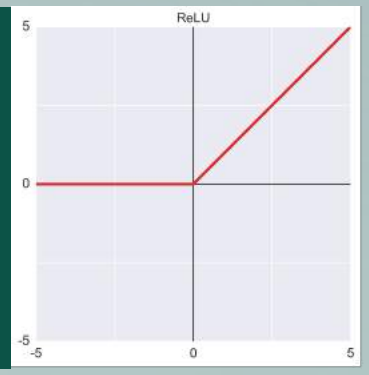


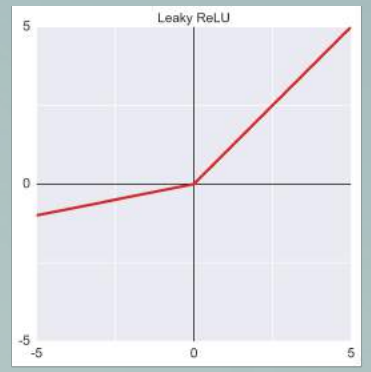
Activation Functions

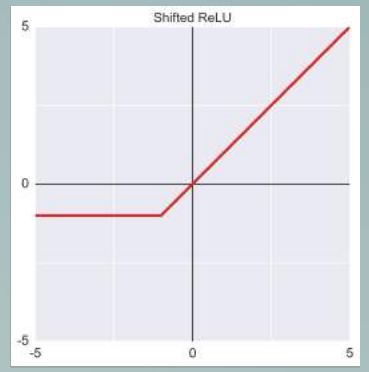
An activation function takes a floating-point number as input and returns a new floating-point number as output.

In practice, we apply the same activation function to all neurons in each layer.

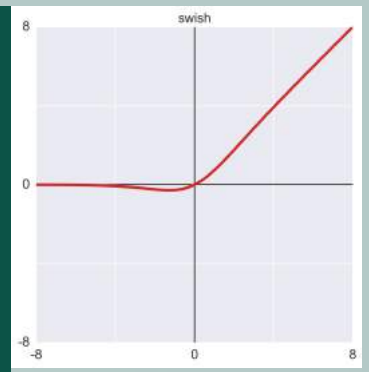
picewise linear

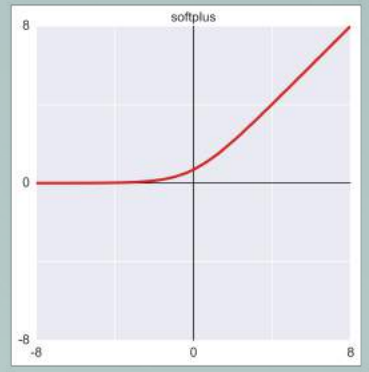


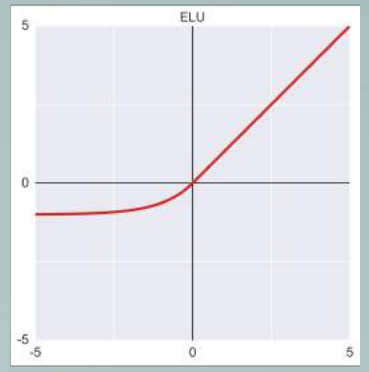


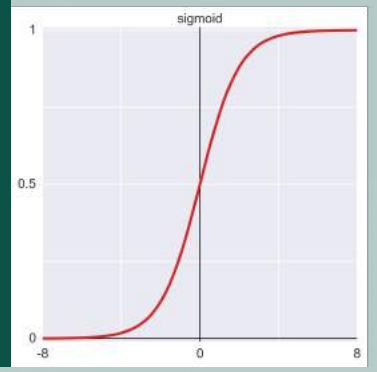


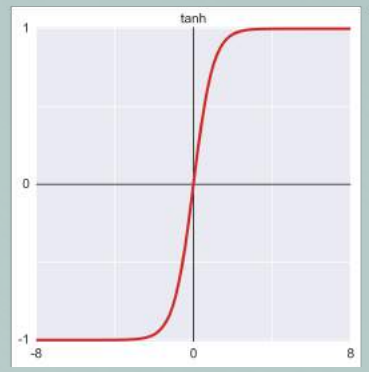
smooth

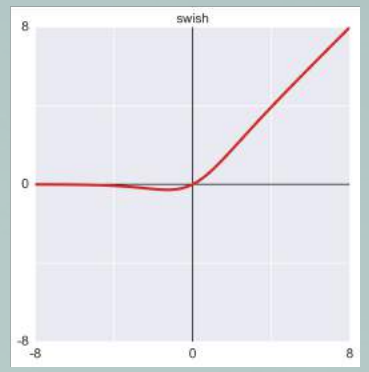










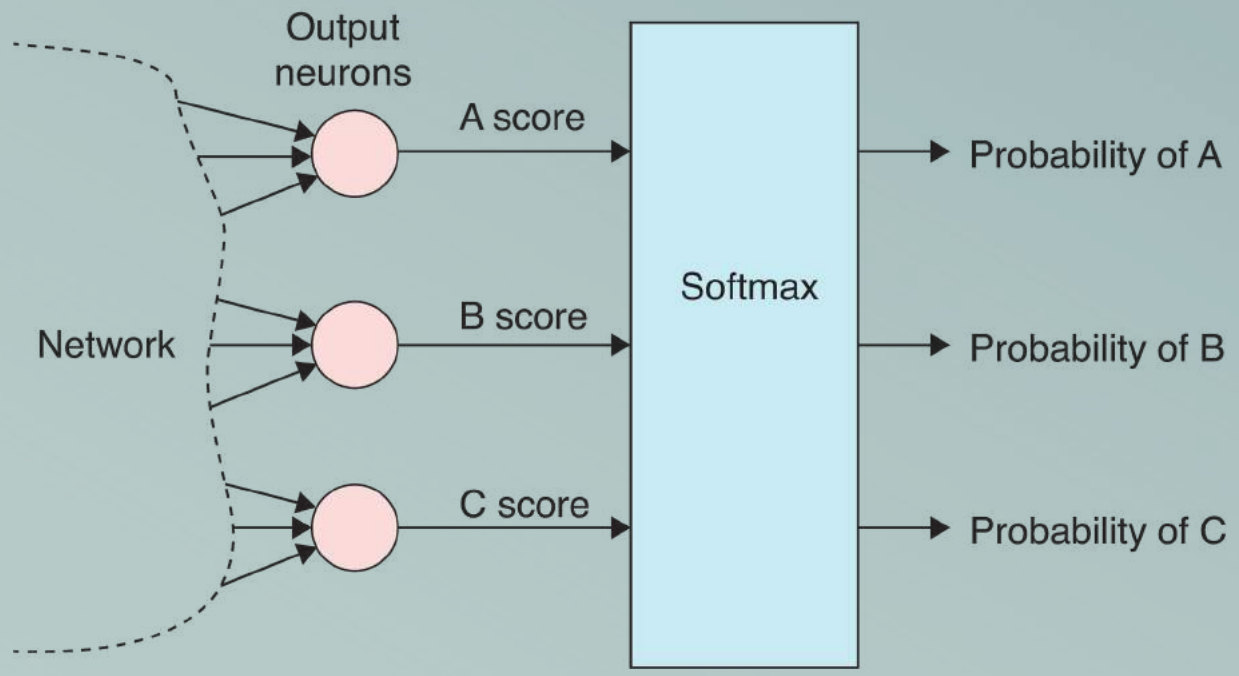


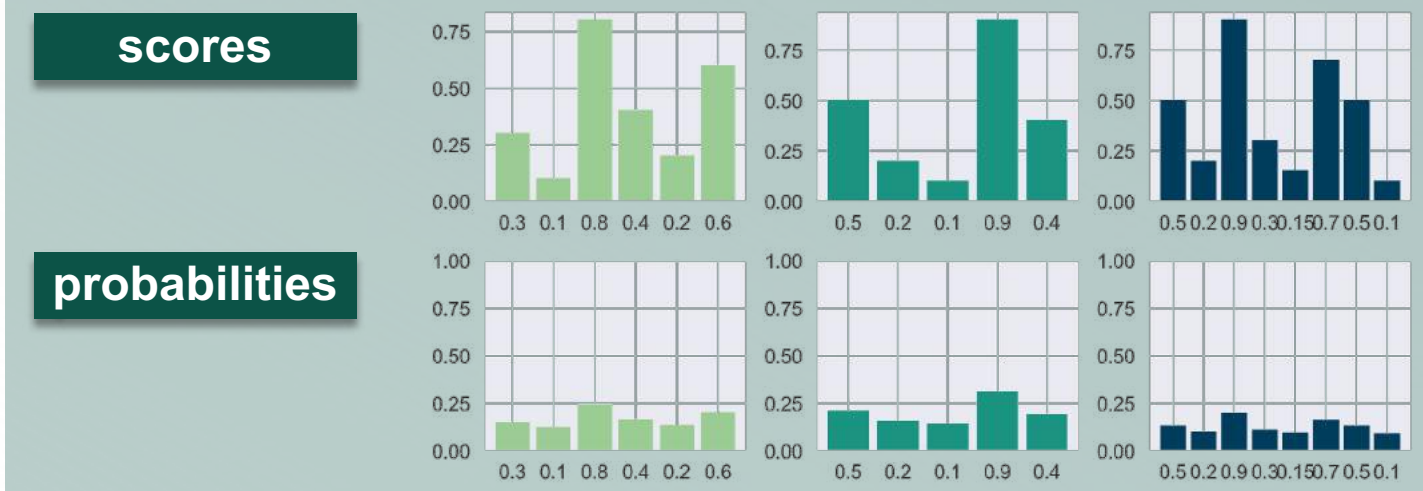
Softmax

The purpose of **softmax** is to turn the raw numbers that come out of a classification network into class probabilities.

Probabilities must satisfy two criteria:

- All values lie between 0 and 1
- All values add up to 1





PRAXIS

Tinker With a Neural Network – Tensor Playground

5

How do neural networks learn?

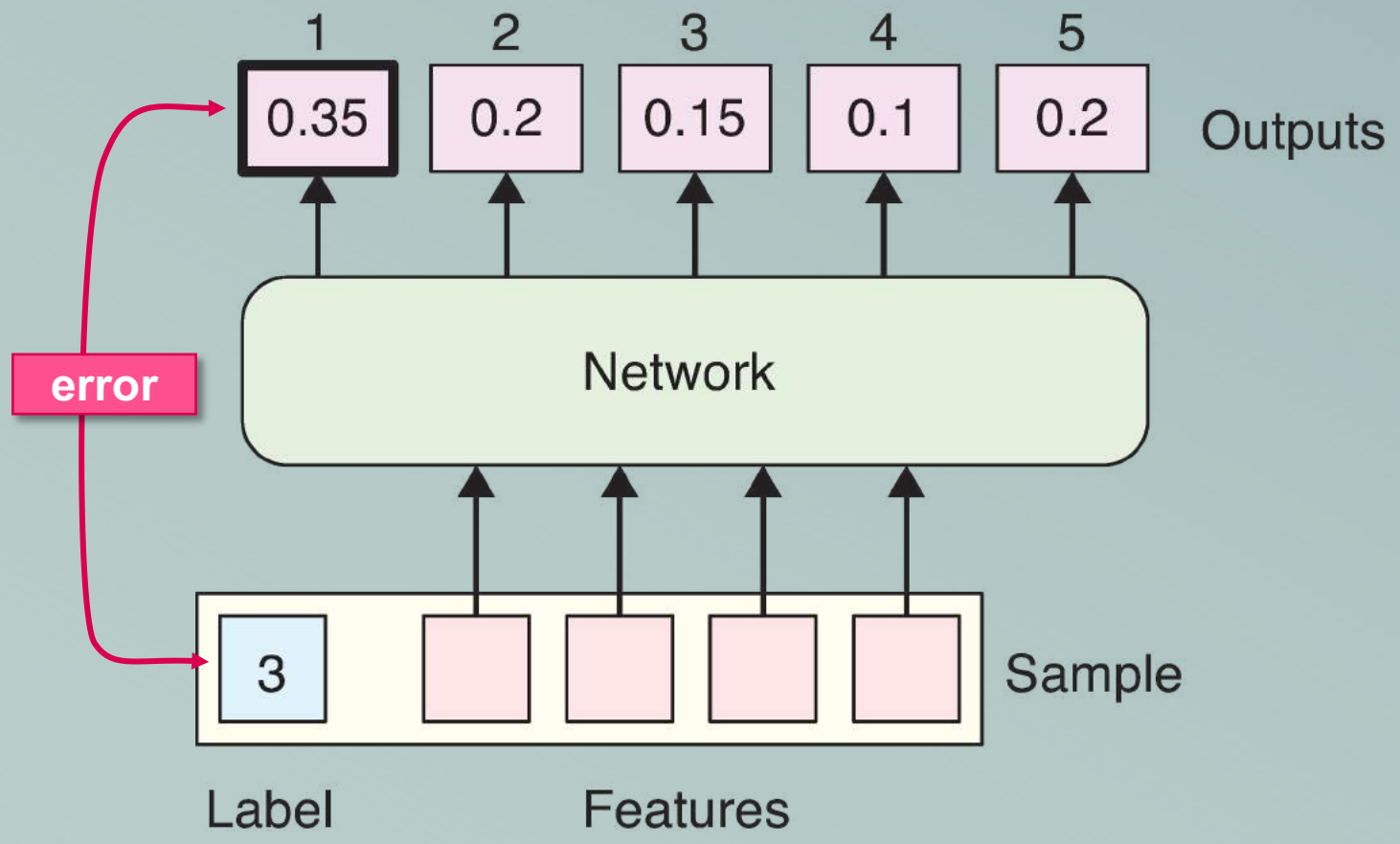
Backpropagation & Gradient Descent

High-level Overview of Training

Suppose we have a classifier that identifies each input as one of five classes. The class that has the largest value is the network's prediction for the current sample.

When a prediction doesn't match the sample's label, we can produce a number to tell us just how wrong this answer is.

We call this number the error score, error, penalty or **loss**.

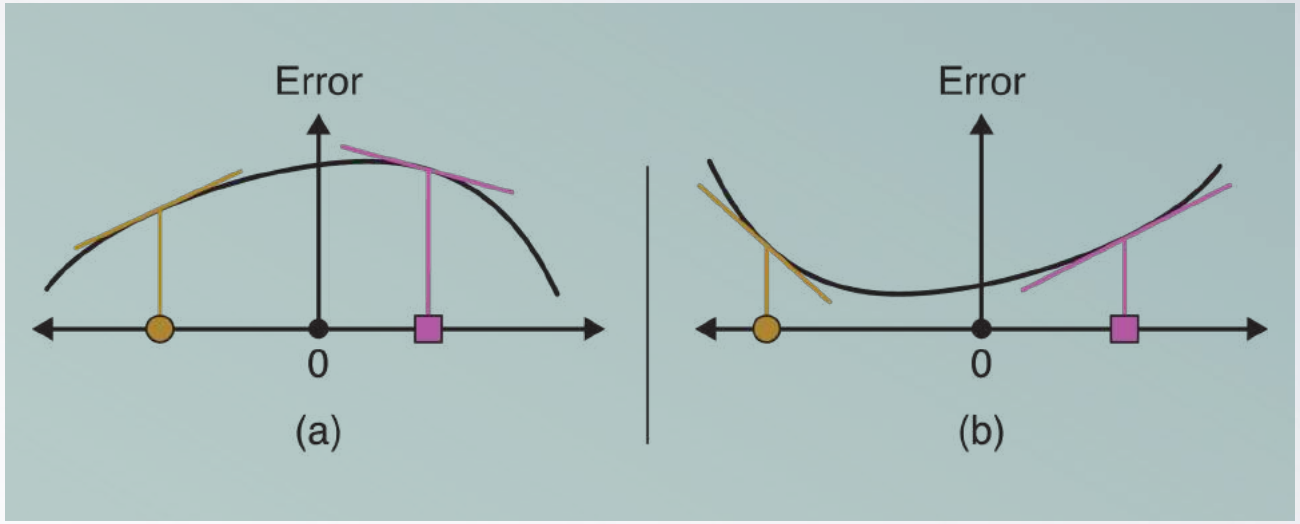


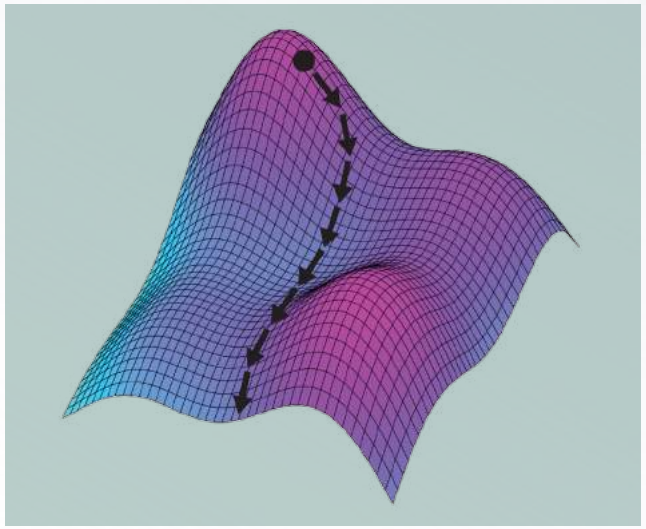
Traversing the error surface

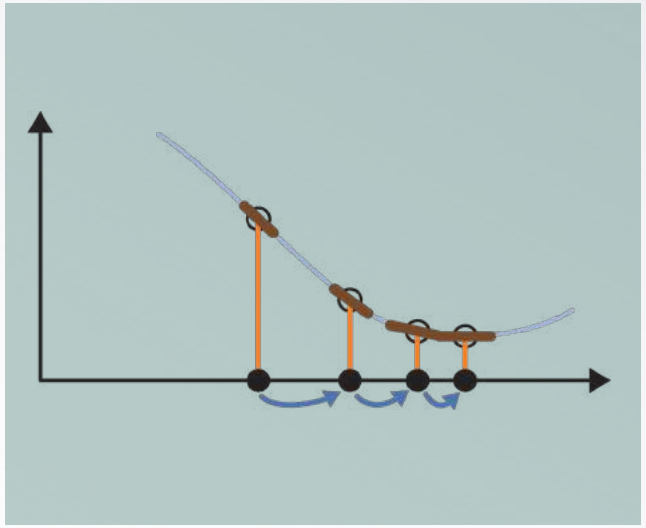
Gradient Descent

The errors form a curve called the **error curve**. In this situation, we can find the gradient (or derivative) of the error at any particular value of the weight by finding the slope of the error curve above that weight.

The error curve for every weight in the network is different because every weight has a different effect on the final error.





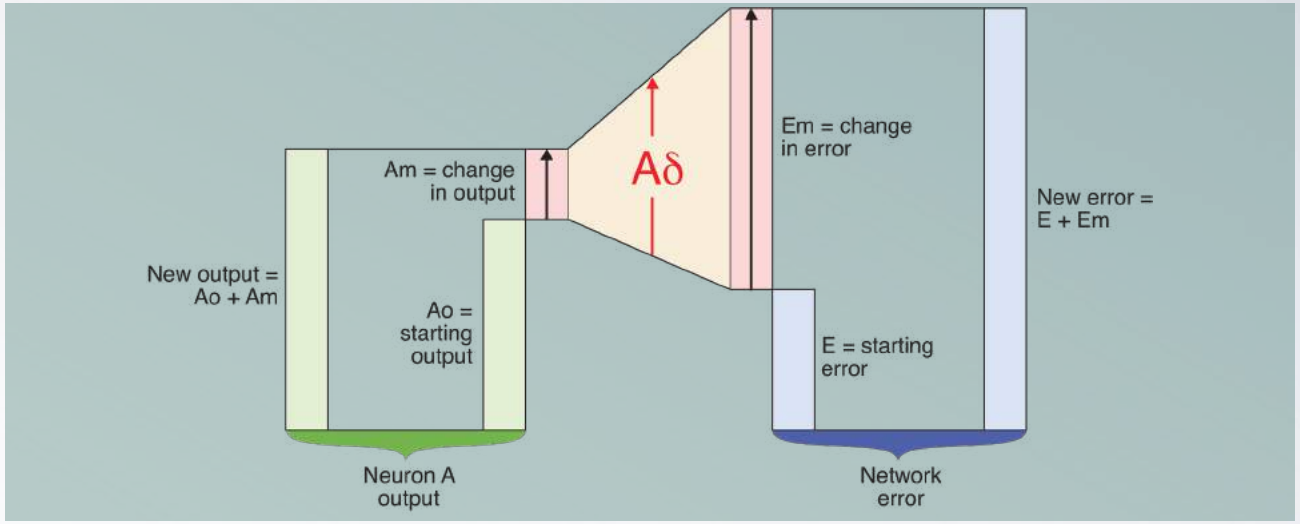


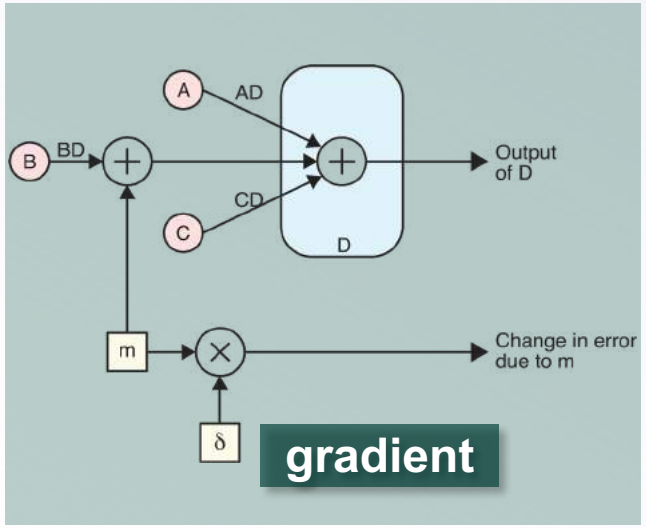
Adjust weights to lower the network's error

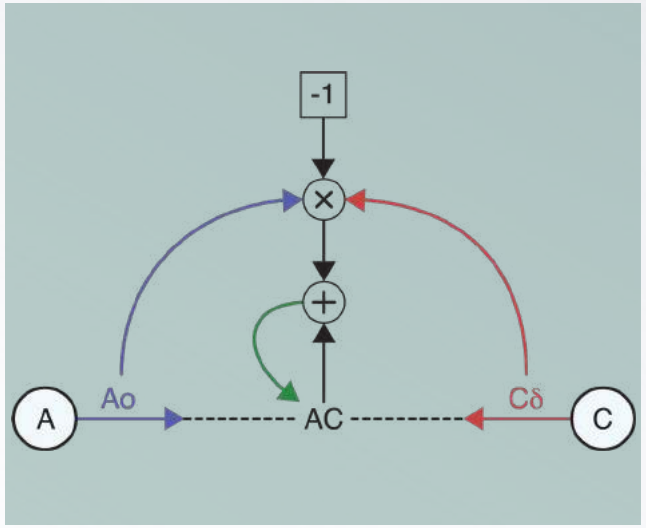
Backprop

During **backpropagation** or backprop, we visit each neuron where we calculate and store a number that is related to the network's error.

Once we have this value for every neuron, we use it to update every weight coming into that neuron. This second step is called the update step or the **optimization** step.

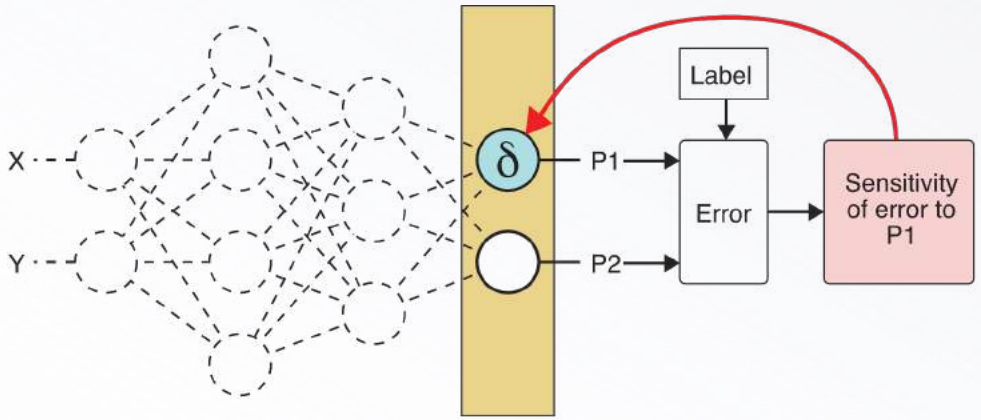


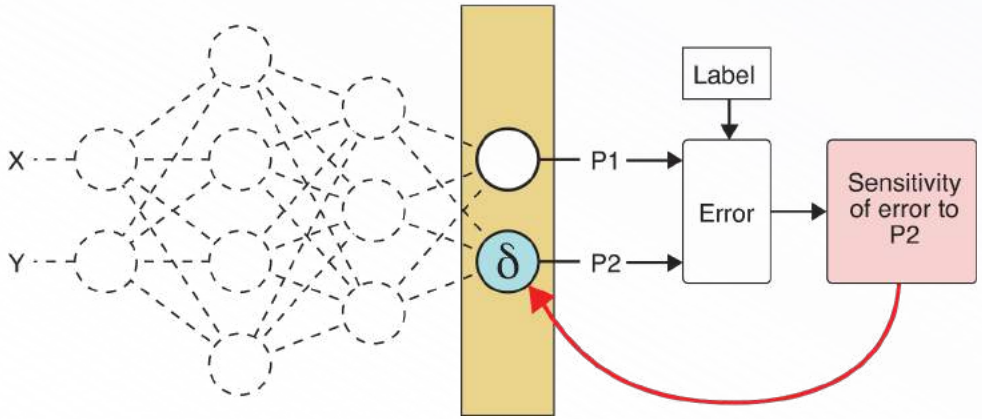


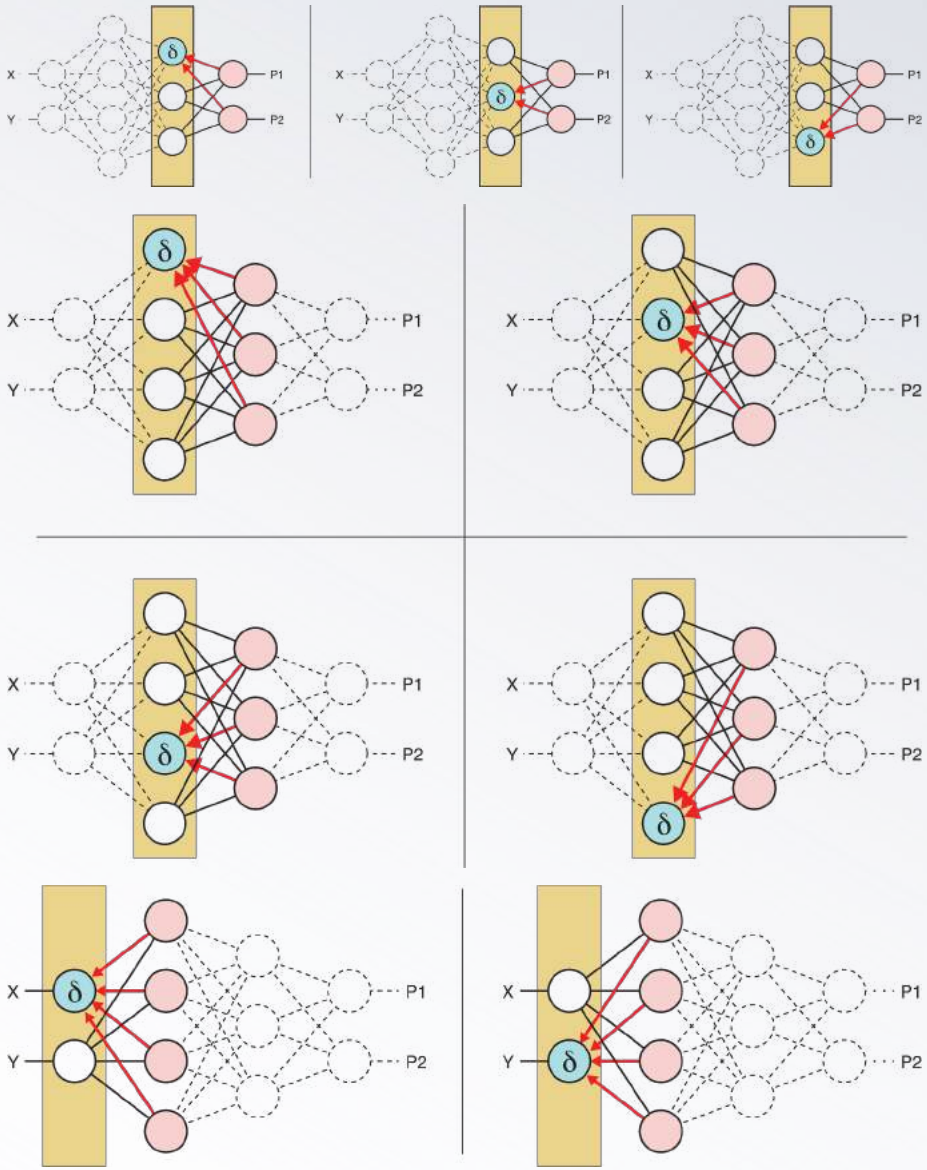


Backprop on a Larger Network

Propagating the gradient backward



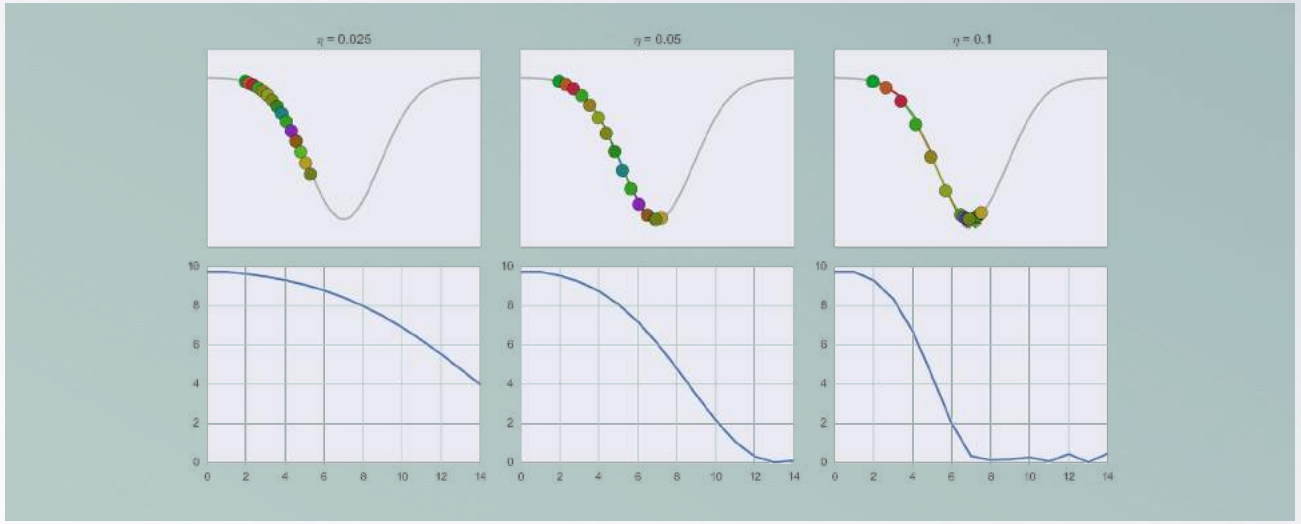


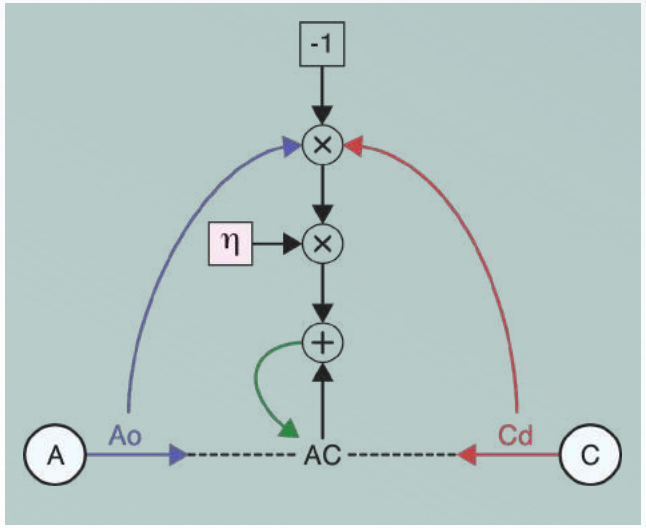


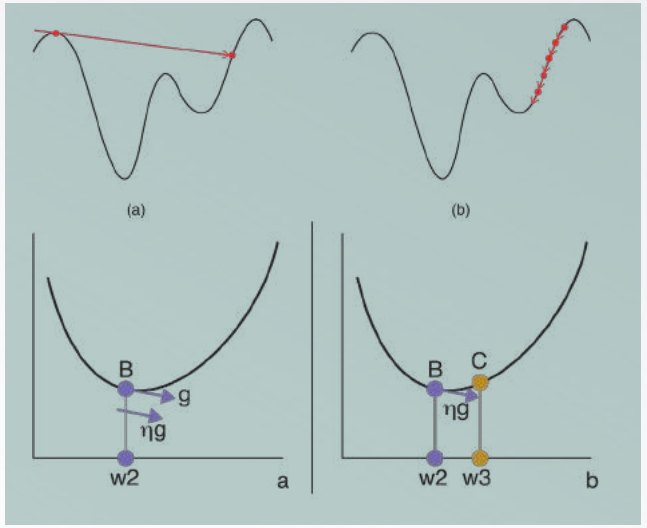
Controlling the step size

The Learning Rate

In practice, we control the amount of change to the weights during every update with a **hyperparameter** called the **learning rate**, usually symbolized by η (eta). This is a number between 0 and 1.



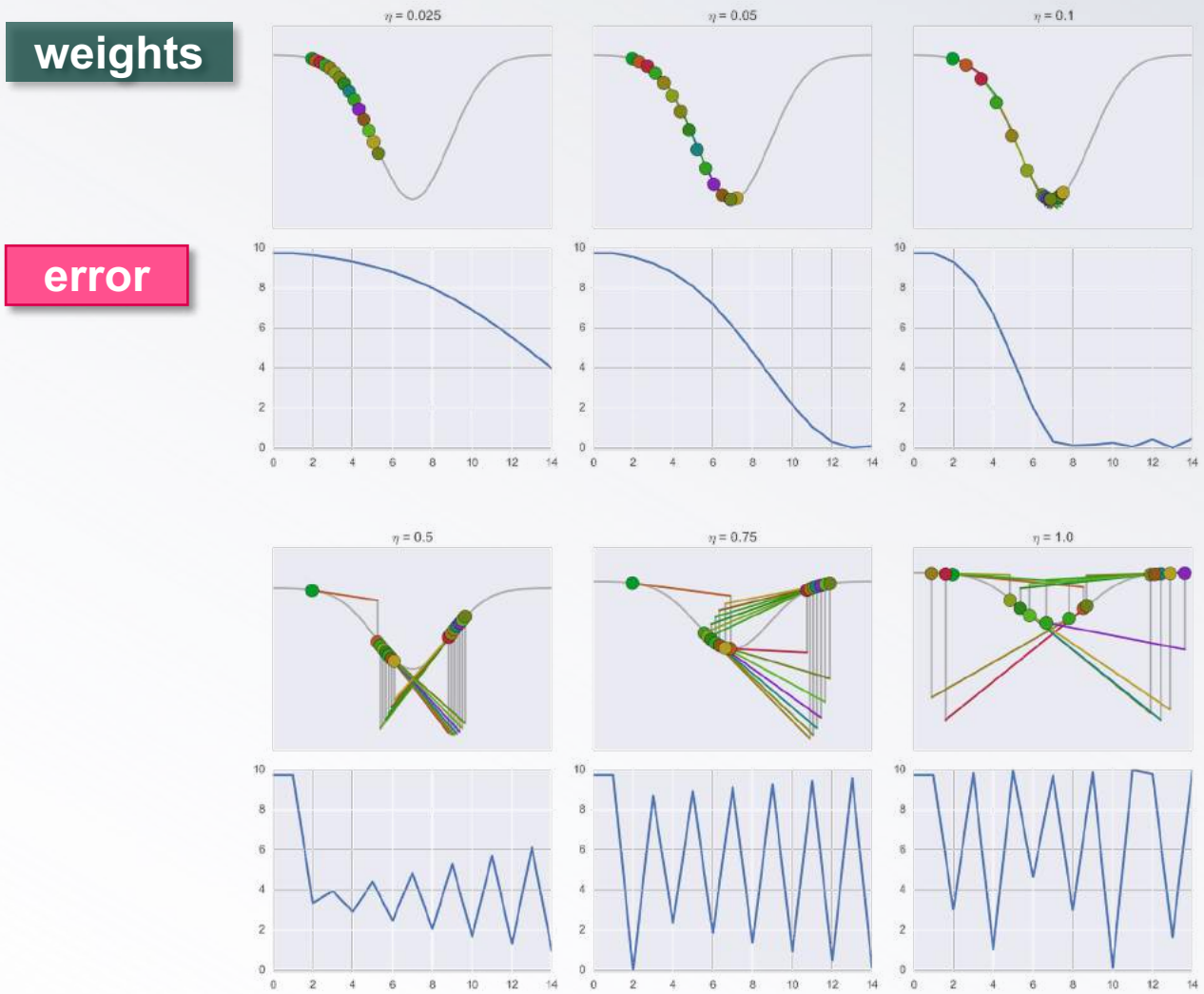




Constant learning rate

Adjusting the Learning Rate

When using gradient descent for learning, the critical parameter is the learning rate. This is often a value in the range 0.01 to 0.0001. Larger values lead to faster learning, but they can lead us to miss valleys by jumping right over them. Smaller values lead to slower learning and can find narrow valleys, but they can also get stuck.



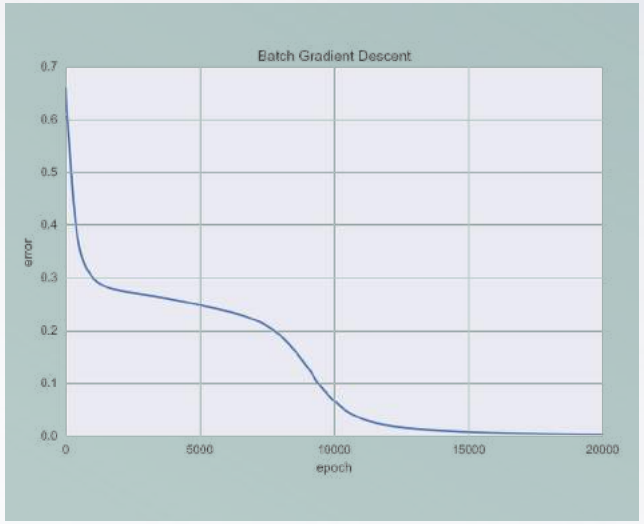
One, Multiple or All!

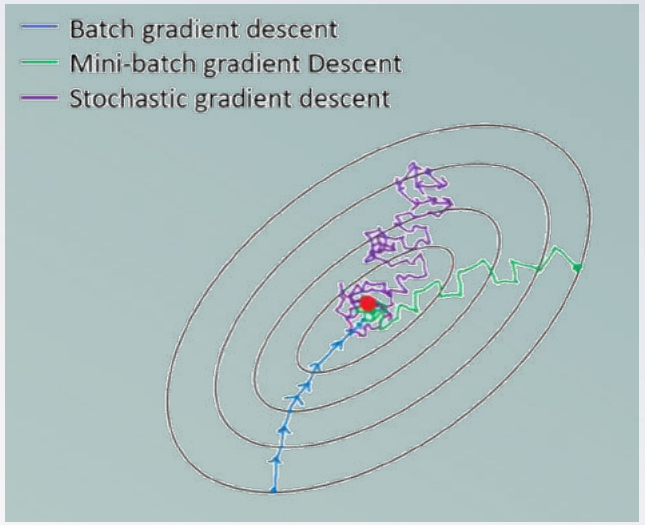
Updating Strategies

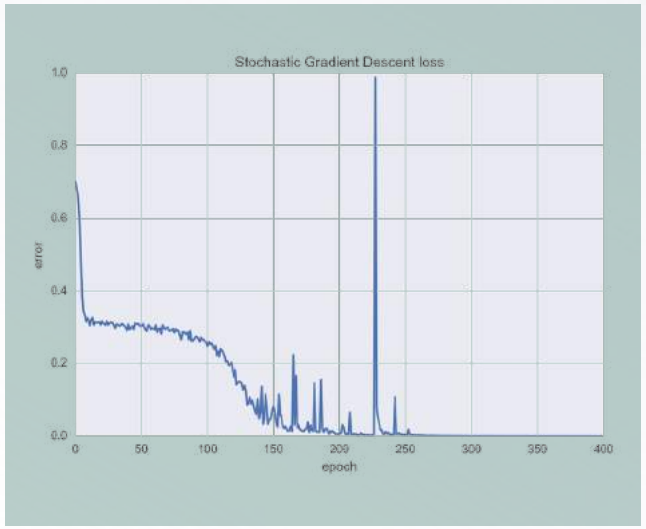
Batch gradient descent updates the weights just once per epoch. We run the entire training set through our system, accumulating the errors. Then we update all of the weights once.

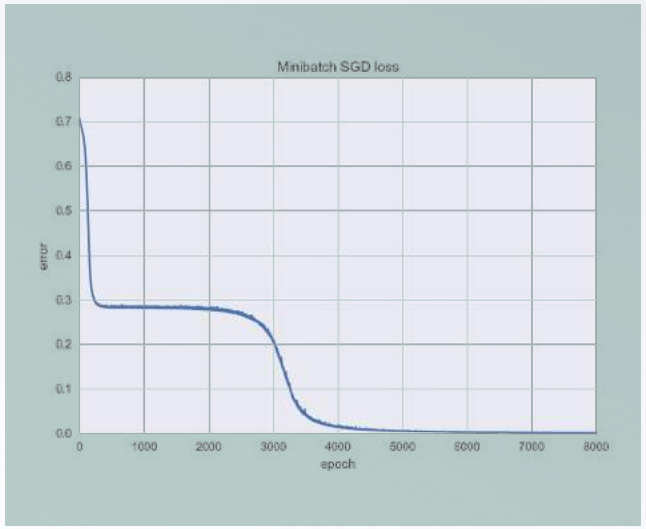
Stochastic gradient descent (SGD) updates the weights after every sample. It‘s stochastic because the training samples are presented in random order.

Mini-batch gradient descent (mini-batch SGD) is a compromise between the two extremes, updating the weights after some fixed number of samples has been evaluated.









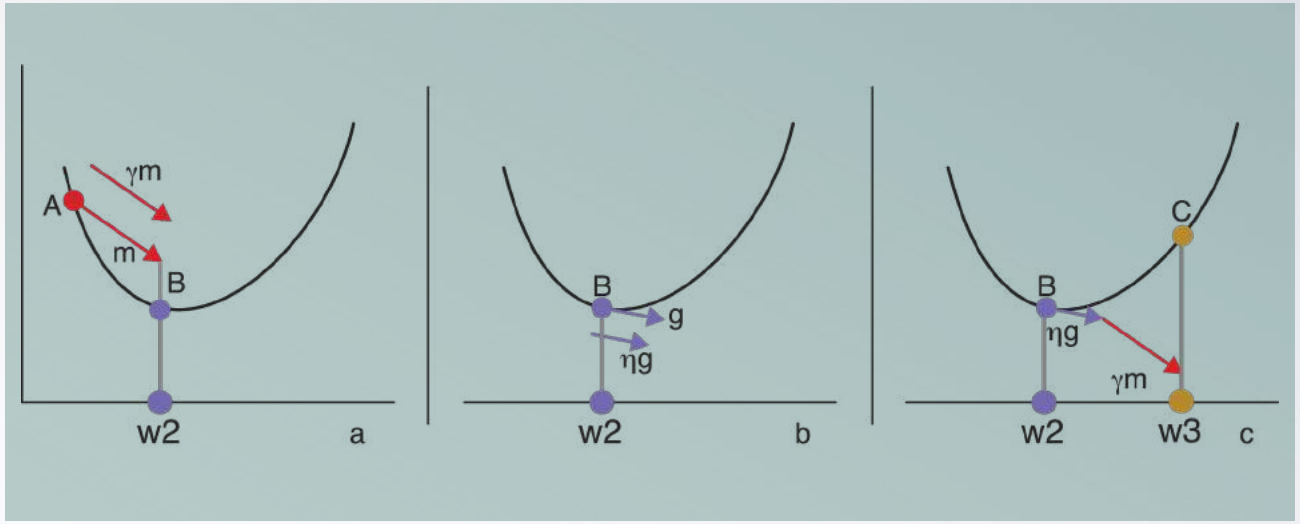
Source: https://medium.com/analytics-vidhya/gradient-descent-vs-stochastic-gd-vs-mini-batch-sgd-fbd3a2cb4ba4

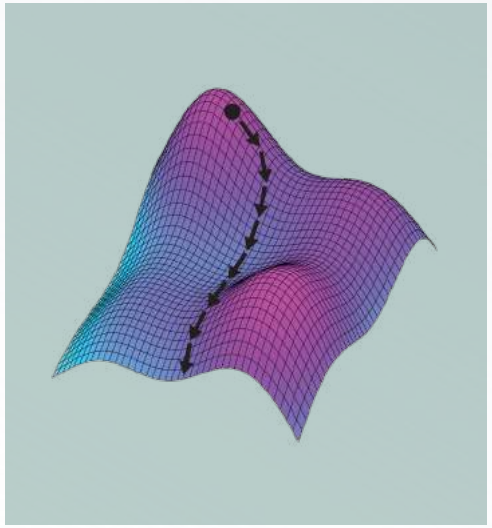
Momentum

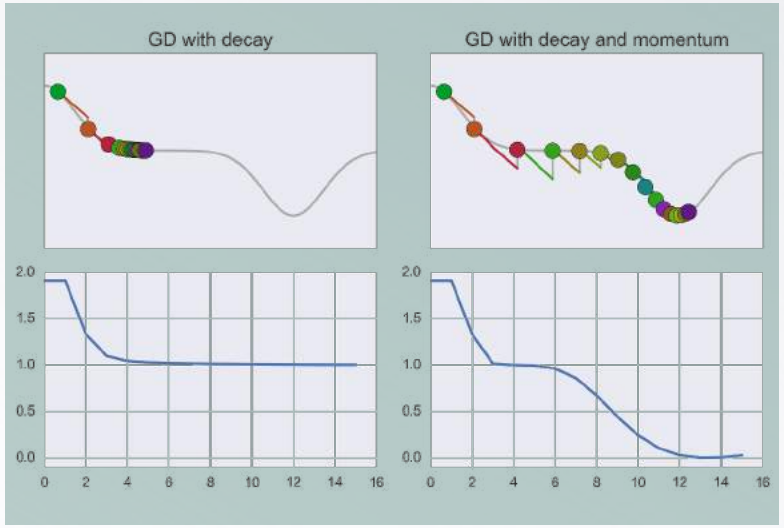
Gradient Descent Variations

The idea of **momentum** is closely related to the concept of **inertia**, which describes the resistance of an object to a change in its motion.

If a ball's motion was determined strictly by the gradient, when it hit the plateau, it would stop. But the ball's momentum keeps it rolling onward.







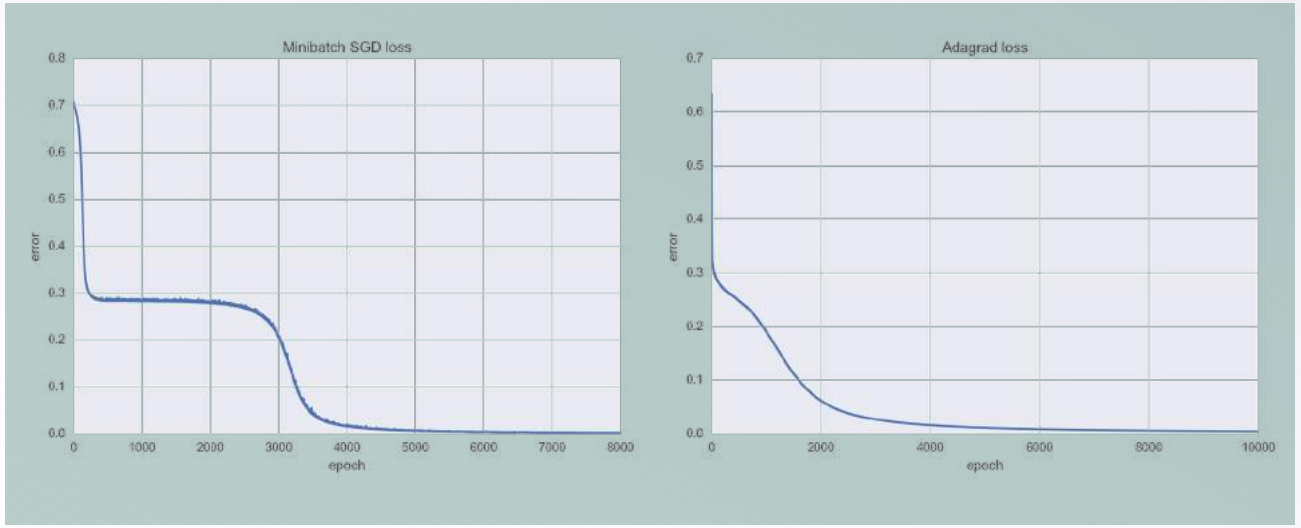
Adaptive gradient learning

Adagrad

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t$$

G_t is sum of squares of the past gradients w.r.t. to all parameters θ

Adagrad adapts the size of the gradient for each weight. Adagrad performs learning-rate decay on a weight-by-weight basis. For each weight, Adagrad takes the current gradient, squares it, and adds that into a running sum of all previous gradients for that weight.



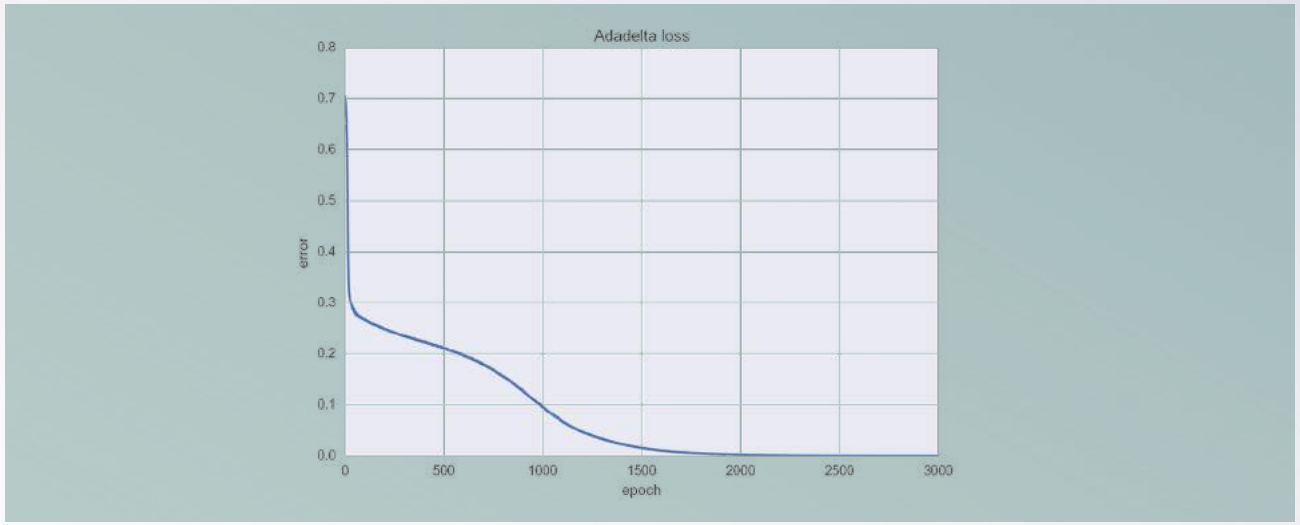
Adaptive gradient learning

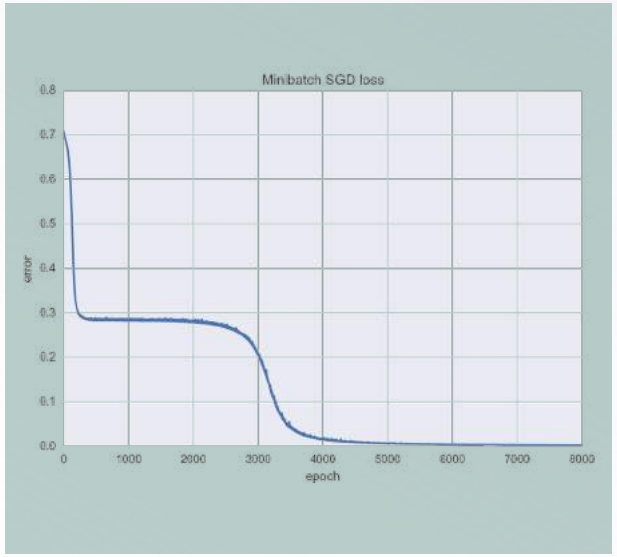
Adadelta and RMSprop

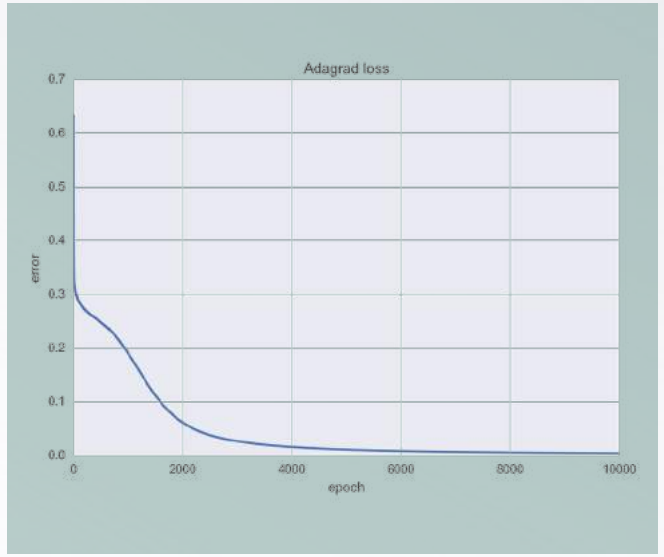
Adagrad gives all the gradients in the sum equal weight when it builds its scaling factor, while Adadelta and RMSprop treat older elements as less important.

Adadelta adaptively changes how much the weights are updated on each step using each one's weighted running sum.

RMSprop is very similar to Adadelta, but it uses a root-mean-squared operation to determine the adjustment that is added to the gradients.



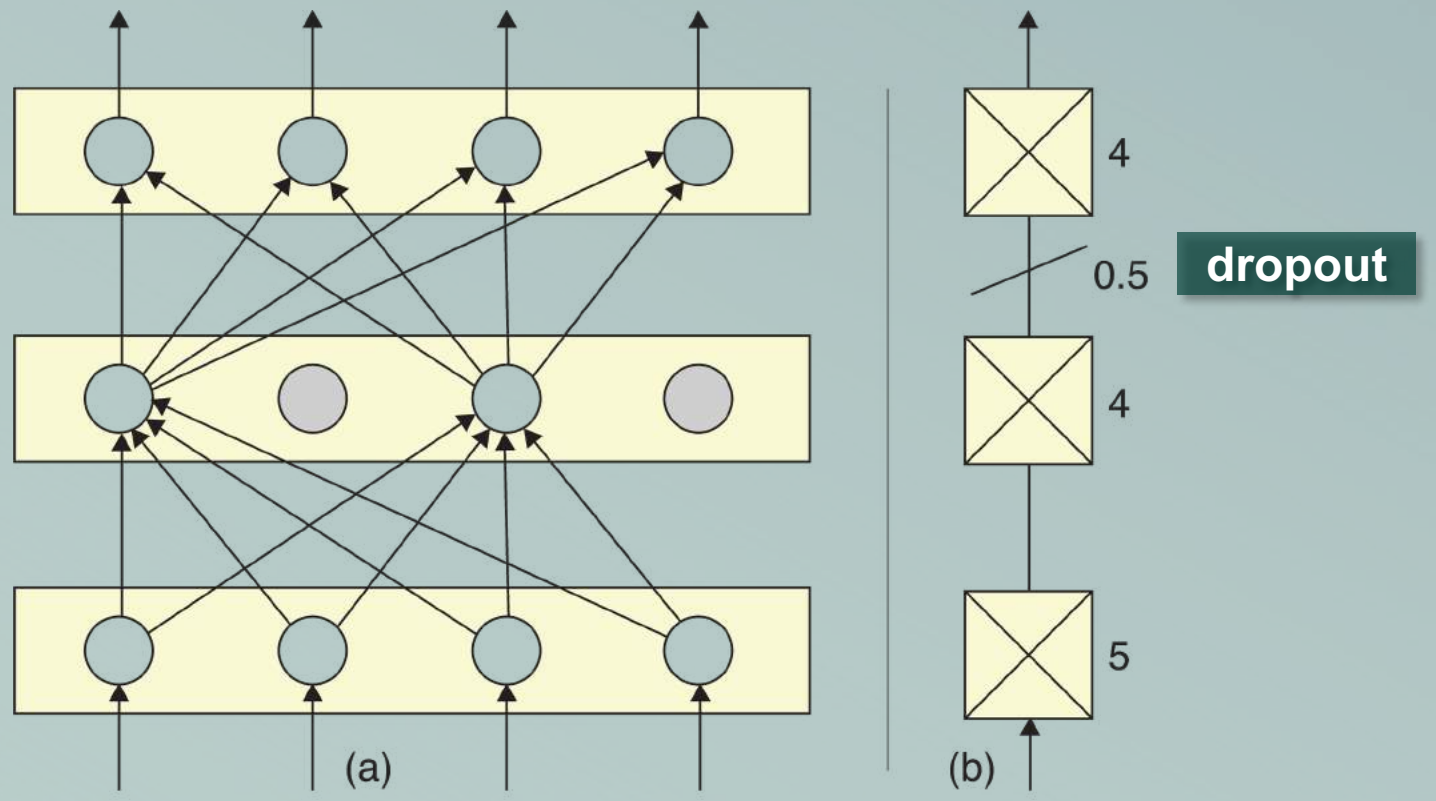




Delay overfitting

Regularization

Dropout is usually applied in the form of a dropout layer. The dropout layer is called an accessory layer or a supplemental layer, because it doesn't do any computation of its own. Dropout temporarily disconnects some of the neurons on the previous layer. Dropout delays overfitting by preventing any neuron from overspecializing and dominating.

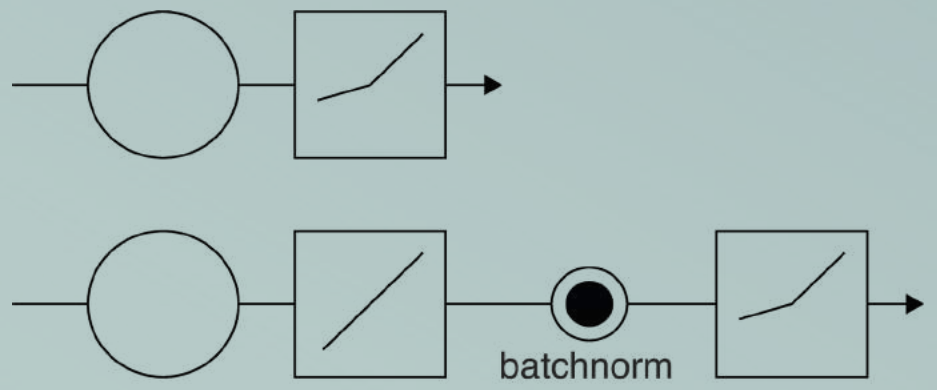


Delay overfitting

Regularization

Batch normalization, or batch norm, modifies the values that come out of a layer before applying the activation function so that the modified values will fall in the region of the activation function where they are affected the most (usually around 0).

Batch norm defers the onset of overfitting, allowing us to train longer.



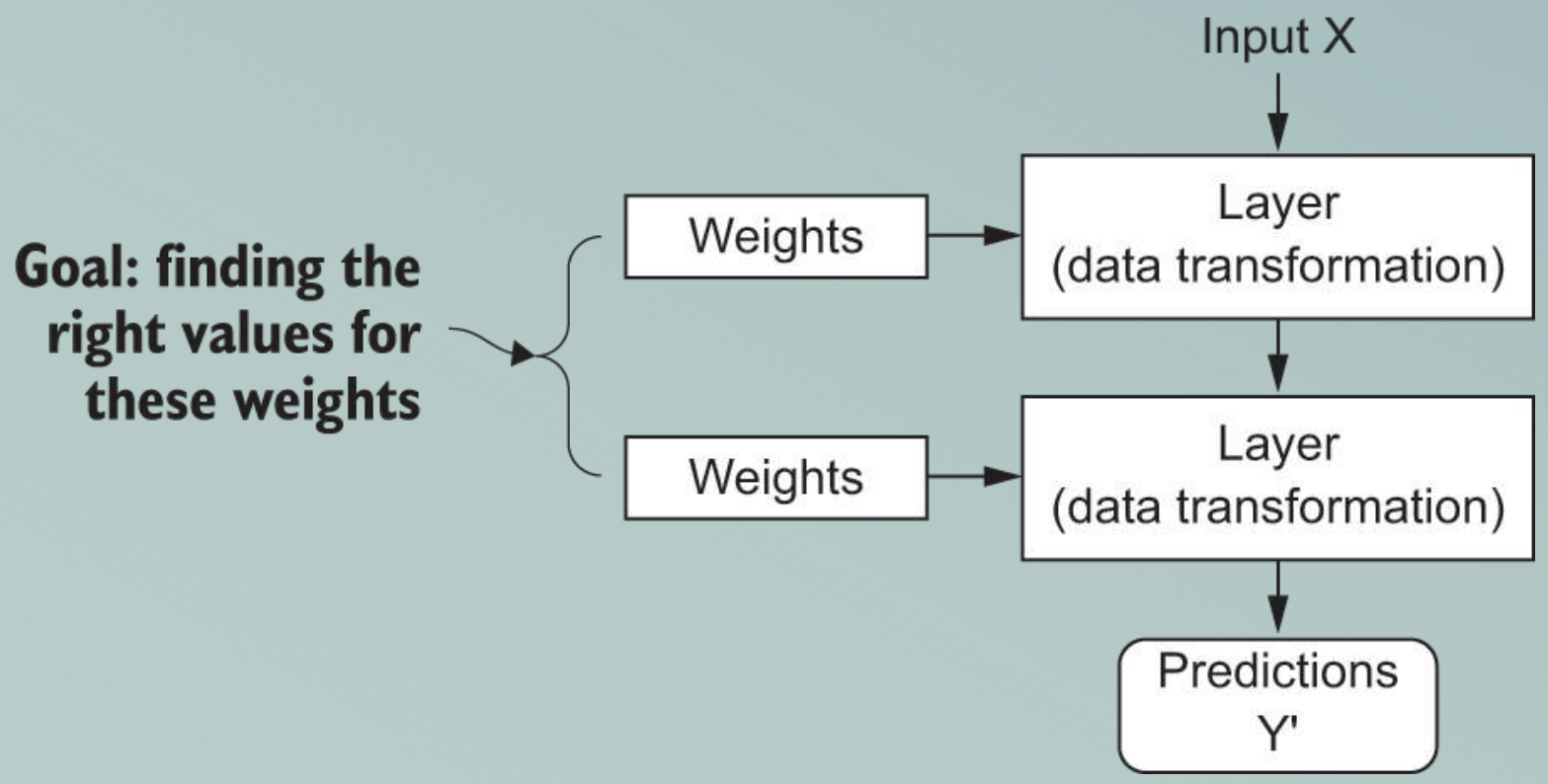
6

Bringing everything together

Understand how Deep Learning works

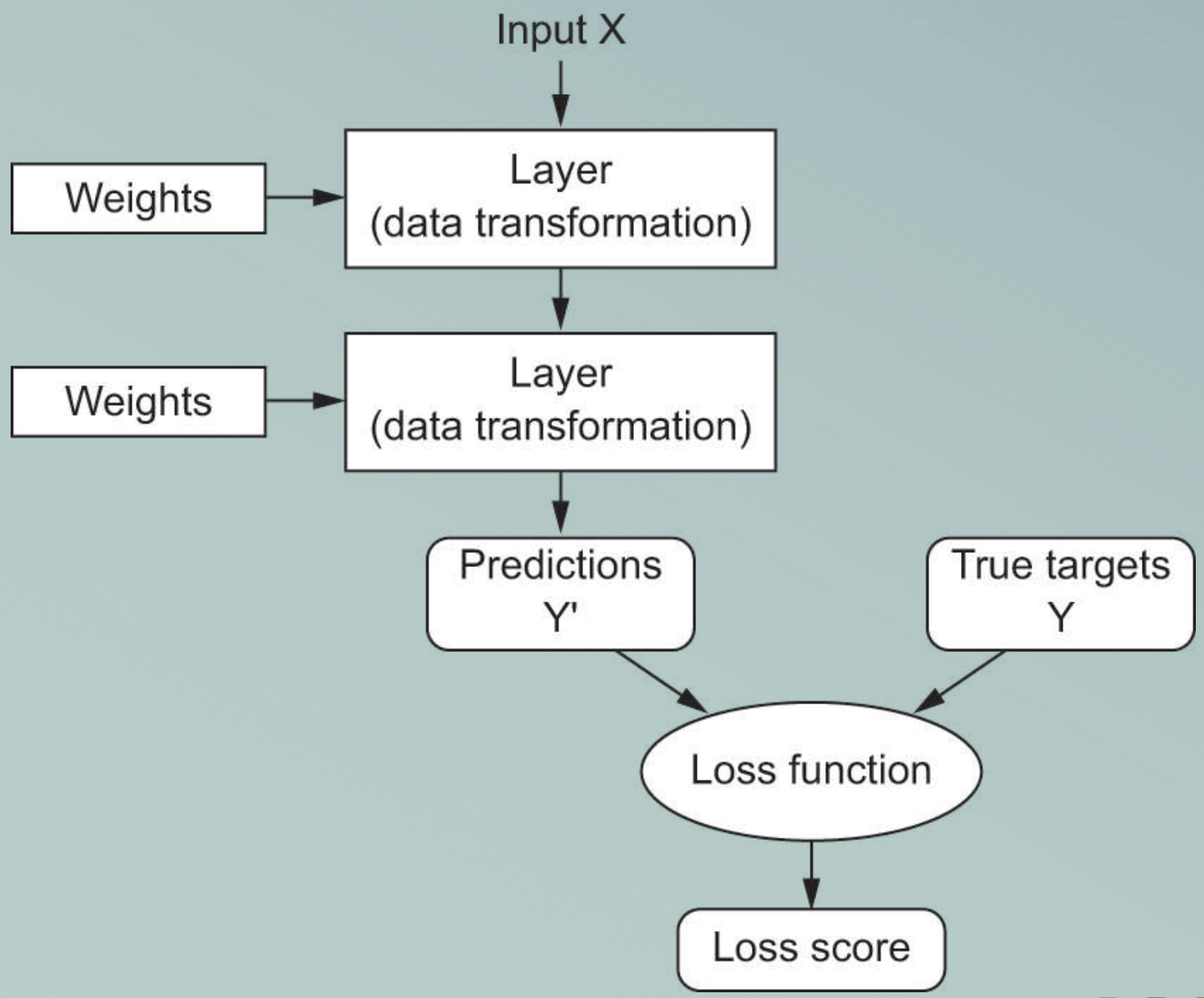
Understanding Deep Learning

Learning means finding a set of values for the **weights** of all layers in a network, such that the network will correctly map example inputs to their associated targets.



Understanding Deep Learning

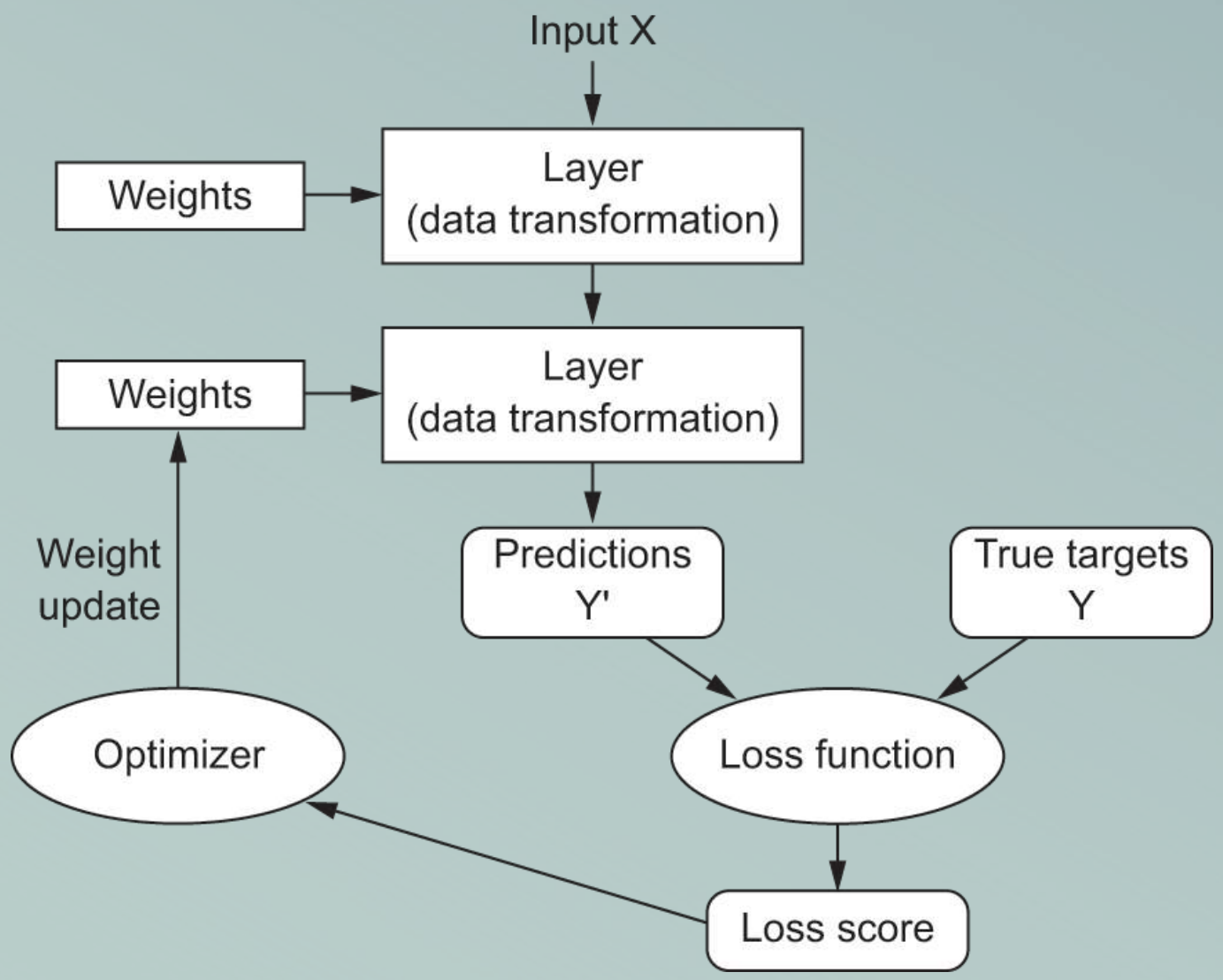
How can you observe the performance of your network? The **loss function**, or cost function, takes the predictions of the network and the true target and computes a distance score, capturing how well the network has done on this specific example.



Understanding Deep Learning

The fundamental trick in DL is to use this score as a feedback signal to adjust the value of the weights a little, in a direction that will lower the loss score for the current example.

This adjustment is the job of the **optimizer**, which implements what's called the Backpropagation algorithm.



Bausteine eines NNs

- Loss → Wie formulieren wir mathematisch, was wir erreichen wollen?
- Optimierer → Welcher Algorithmus minimiert unseren Loss?
- Metric → Welches menschlich verständliche Maß verwenden wir um die Performance unseres Modell einzuschätzen?

Unterschied - Loss und Metrik

Warum optimieren wir nicht nach der Metrik?

- Metrik nicht unbedingt ableitbar
- Metrik ändert sich oft initial wenig
- Metrik kann natürliche Plateaus erreichen

→ Gute Lossfunktionen weisen diese Eigenschaften nicht auf!

How Deep Learning generalizes

It has been hypothesized that the visual hierarchy untangles object manifolds to transform inseparable object manifolds into **linearly separable** ones.

Separability of manifolds depends on numerous factors—numbers of neurons and manifolds, sizes and shapes of the manifolds, target labels, among others.

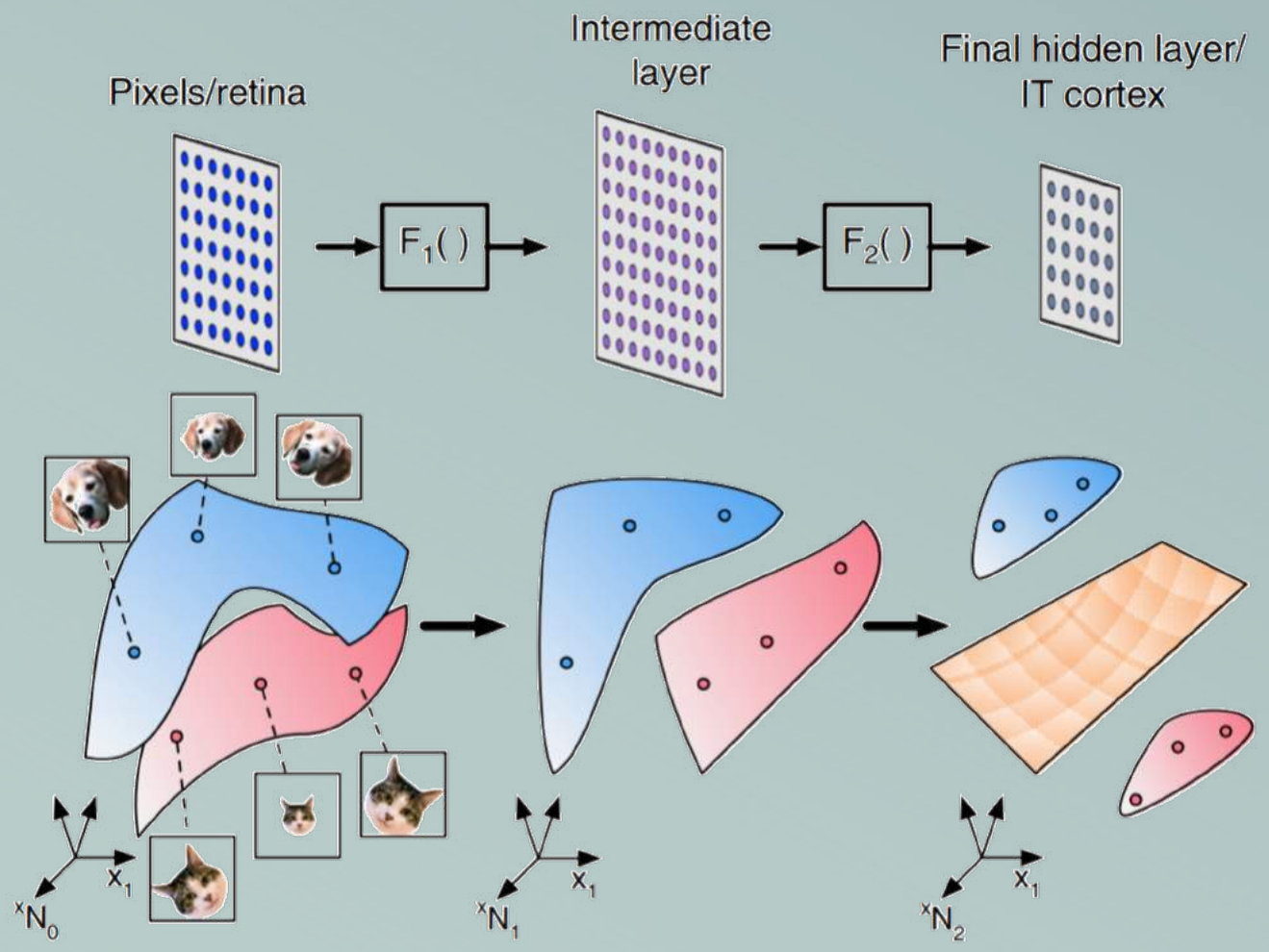
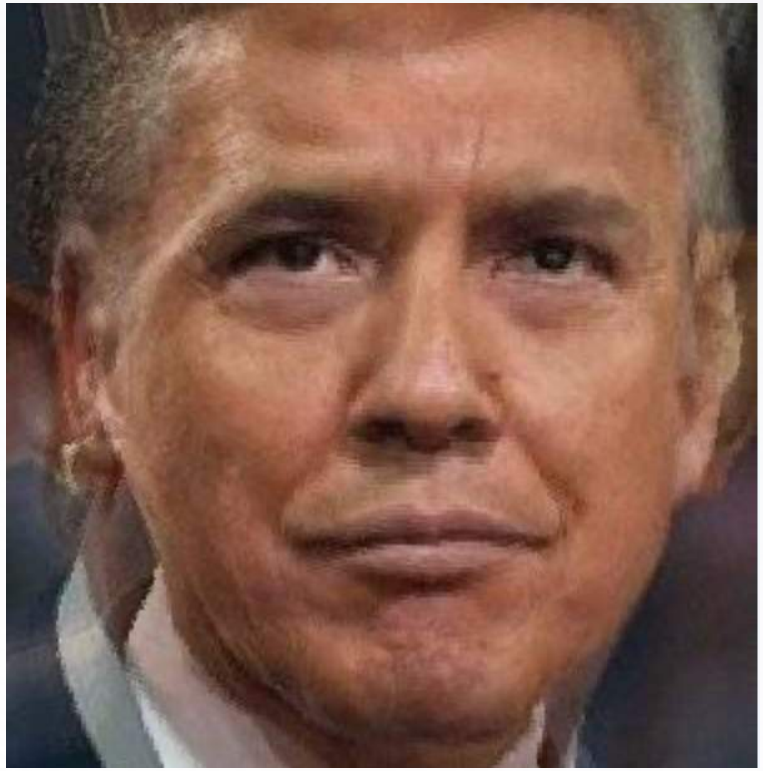


Fig. 1 Changes in the geometry of object manifolds as they are transformed in a deep neural network. Illustration of three layers in a

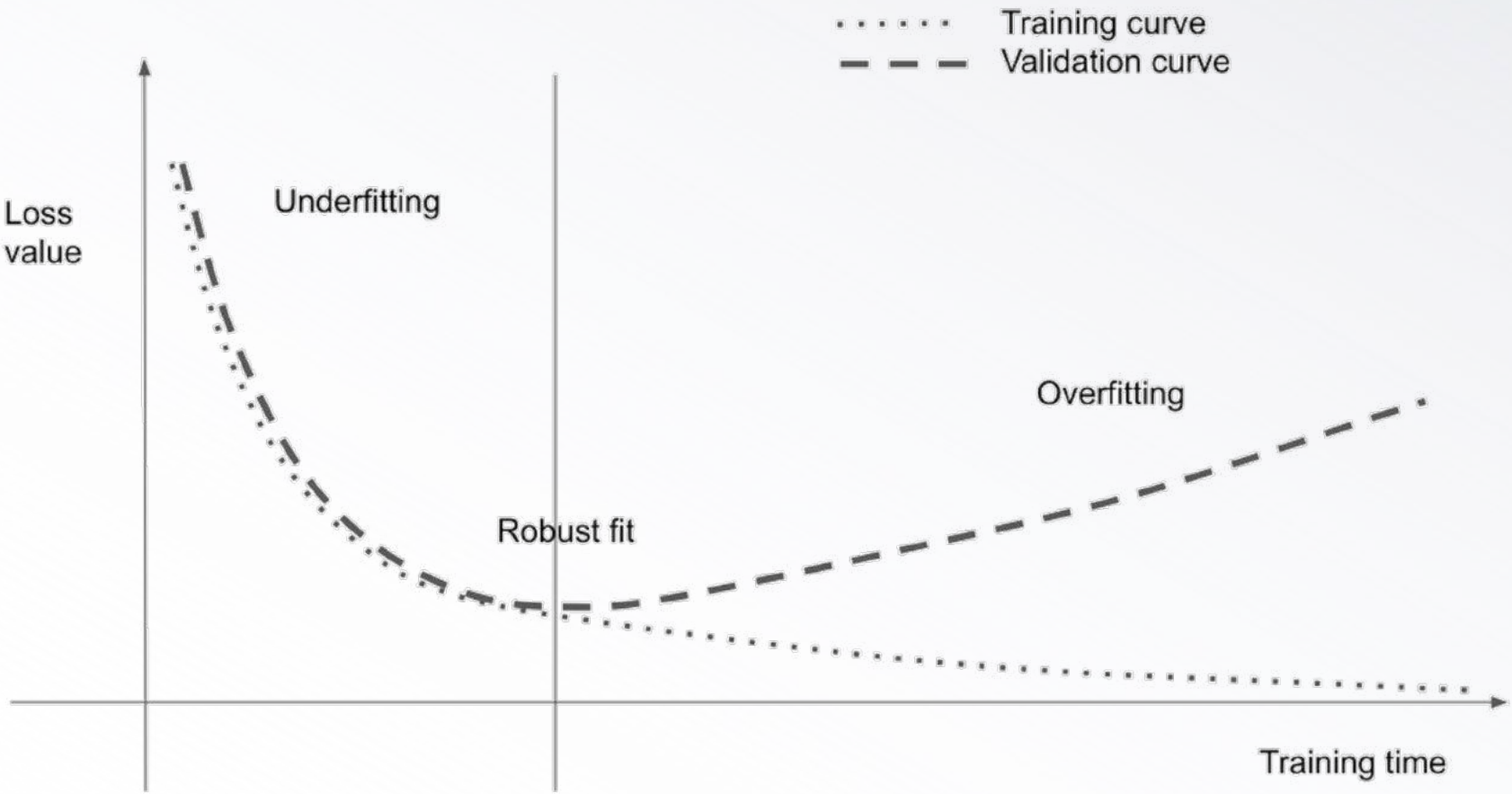
Cohen, U., Chung, S., Lee, D.D. *et al.* Separability and geometry of object manifolds in deep neural networks. *Nat Commun* **11,** 746 (2020). https://doi.org/10.1038/s41467-020-14578-5

Wie generalisiert Deep Learning?

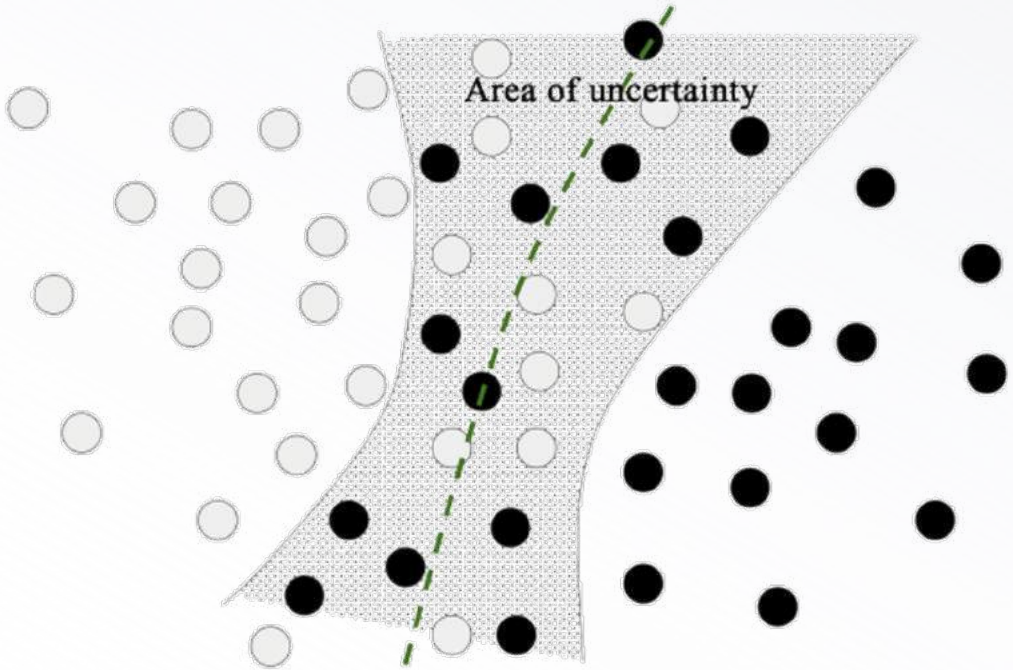
- Deep Learning Modelle intrapolieren innerhalb einer Mannigfaltigkeit
- Eine Klasse bildet eine niedrig dimensionalen, zusammenhängenden Unterraum der von dem Modell modelliert wird.

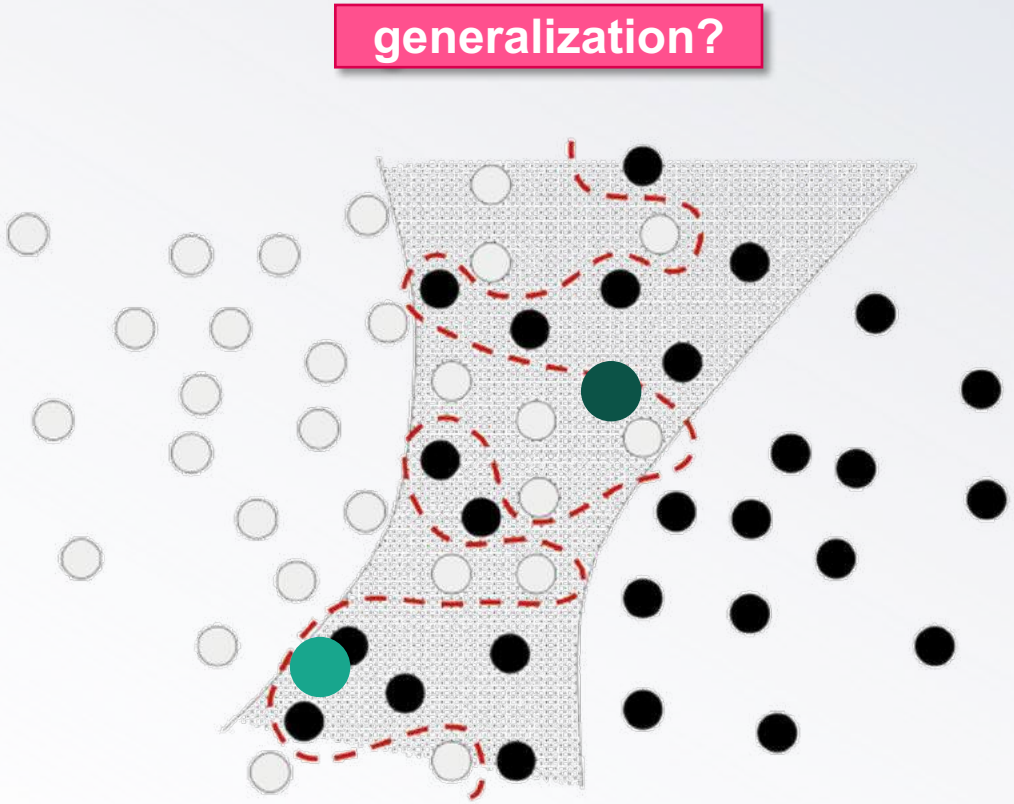


Typische Overfitting Story

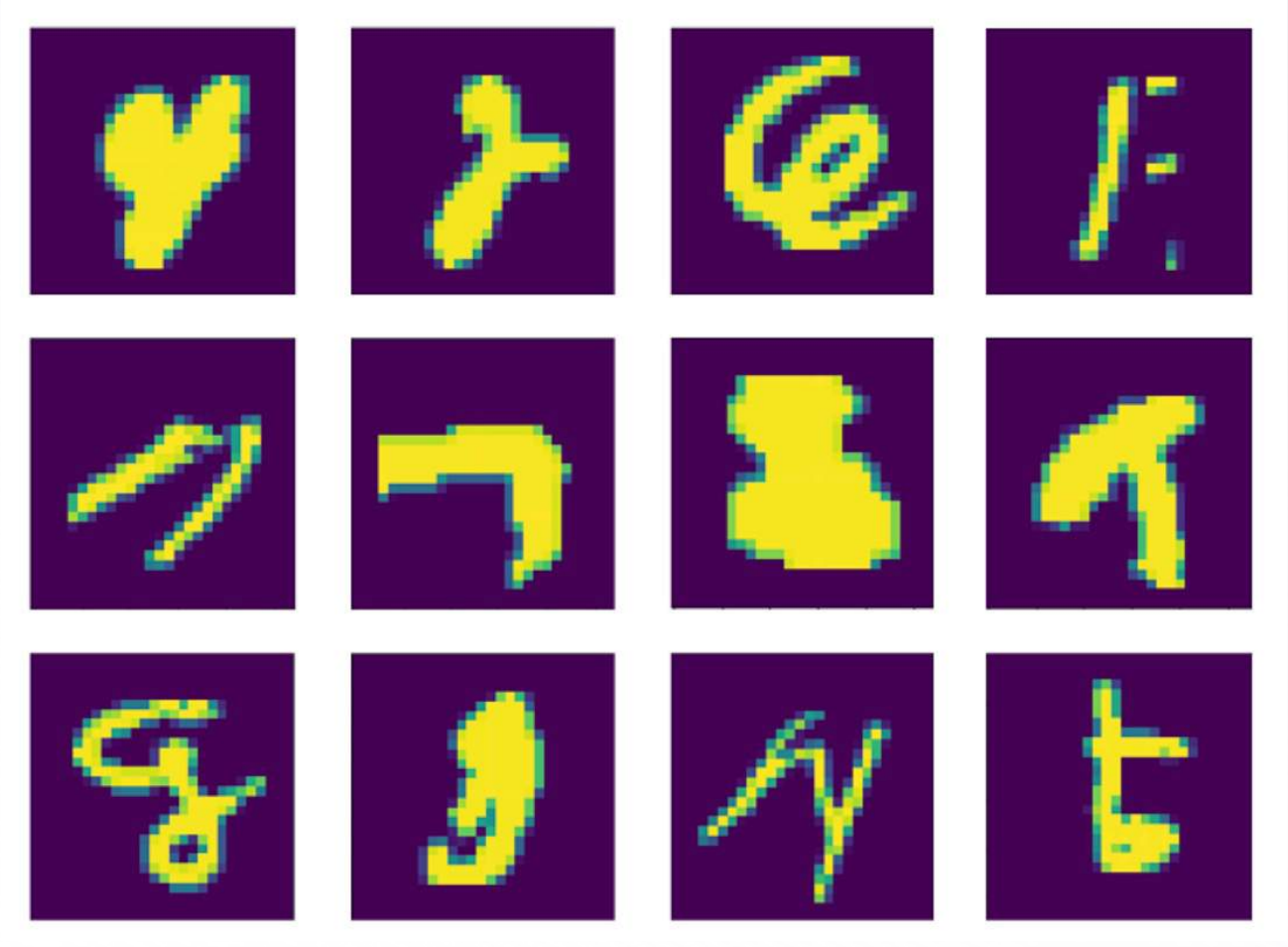


Reality of bad generalization

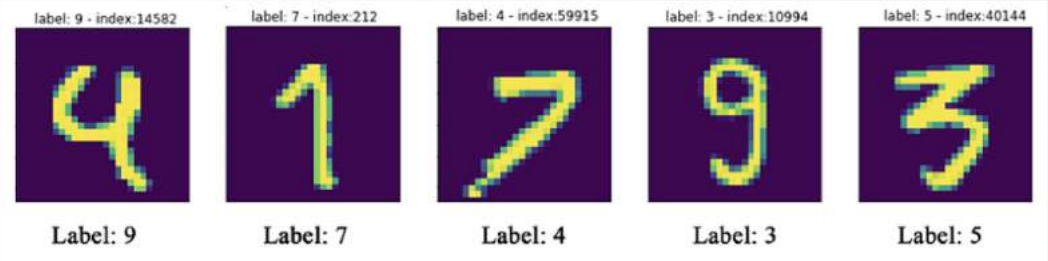




Noise auswendig lernen



- Falsche Labels
- Seltene Features
- Ambiguität
- Rauschen



Einflüsse auf Overfitting

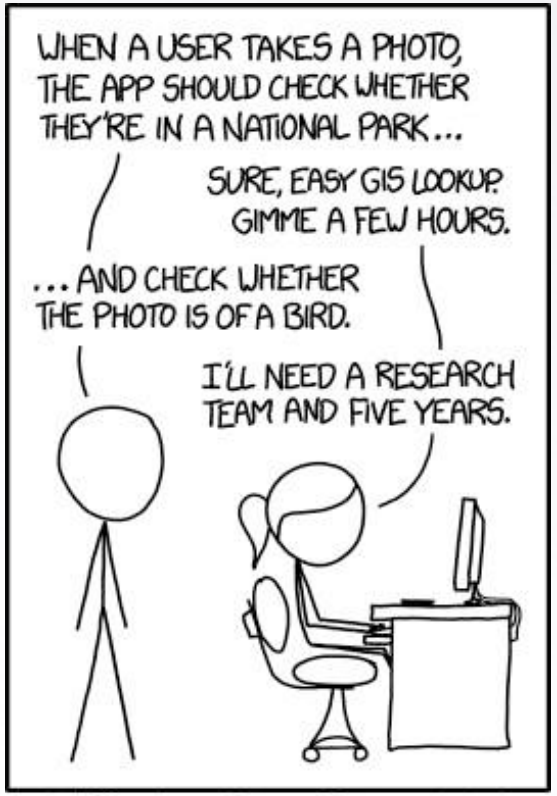
- Learning rate → Saturisierung
- Datenmenge
- Datenkomplexität
- Modelgröße
- Regularisierung

7

Learning hierachical features

Convolutional Neural Networks

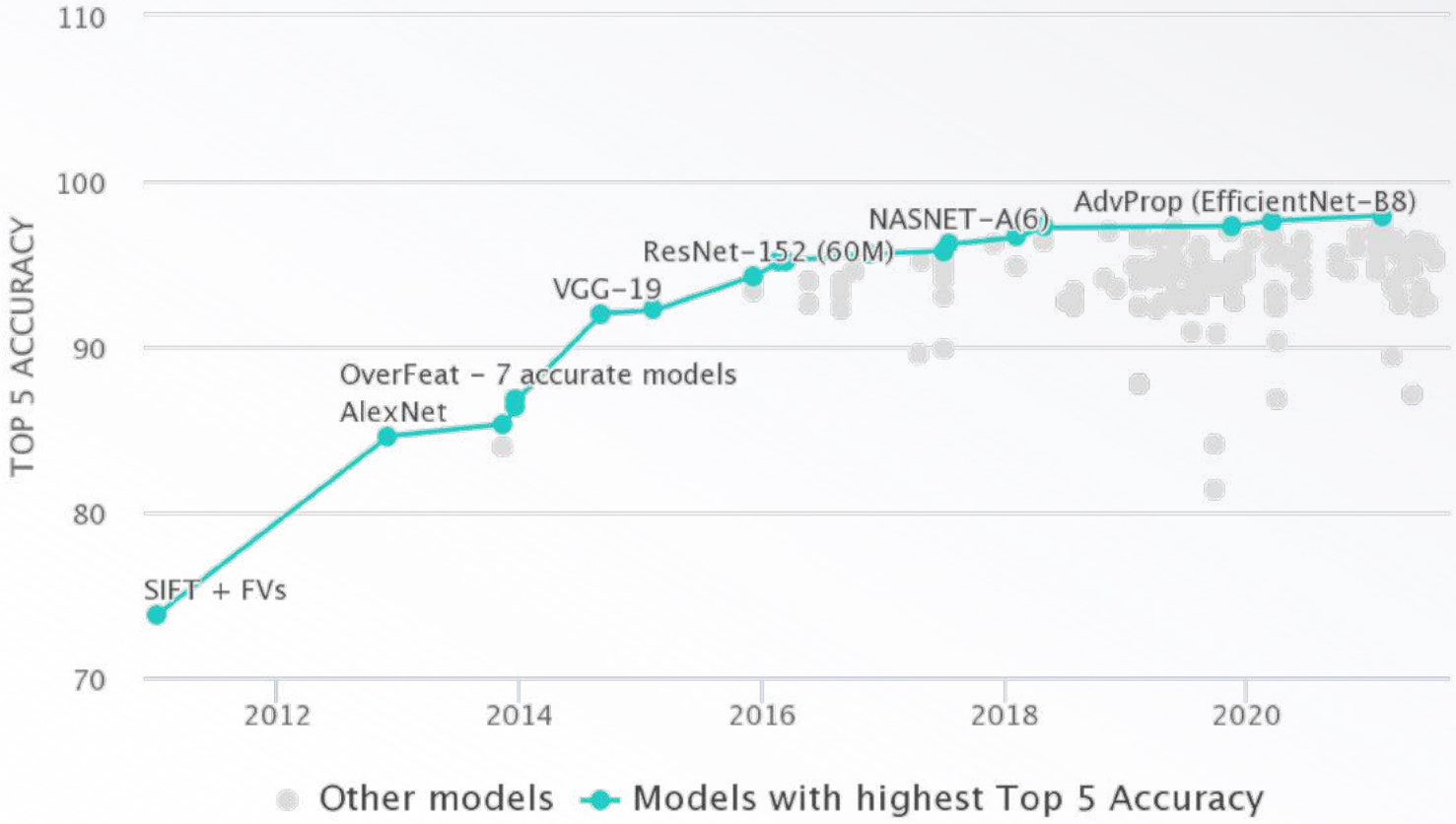
What is hard is hard to tell…

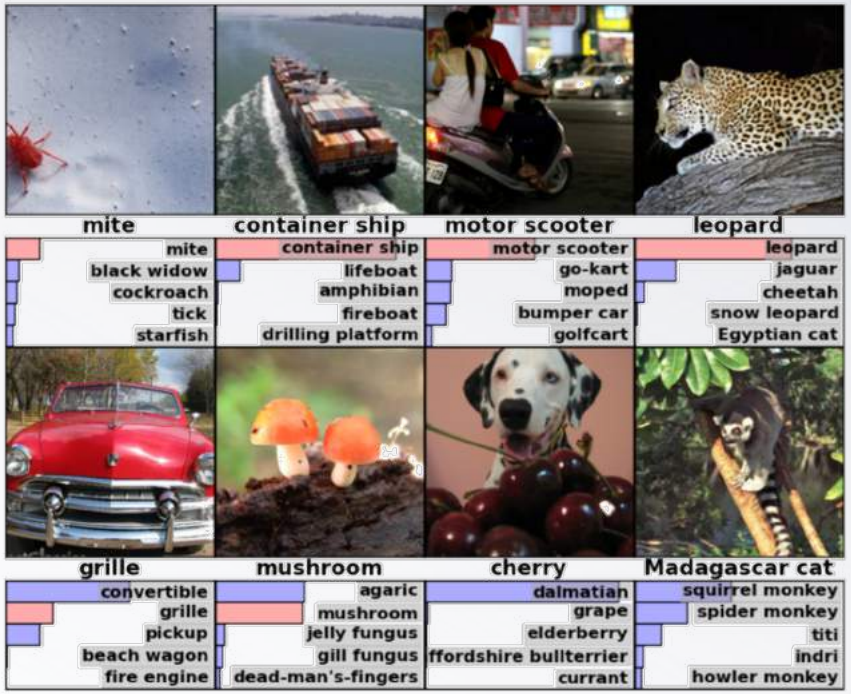


IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Impact of ConvNets

In the 2014 ILSVRC competition a 19-layer CNN achieved a 7.5% top-5 error rate. In the 2015 competition, an ensemble containing 152-layer networks achieved a 3.57% top-5 error rate!)

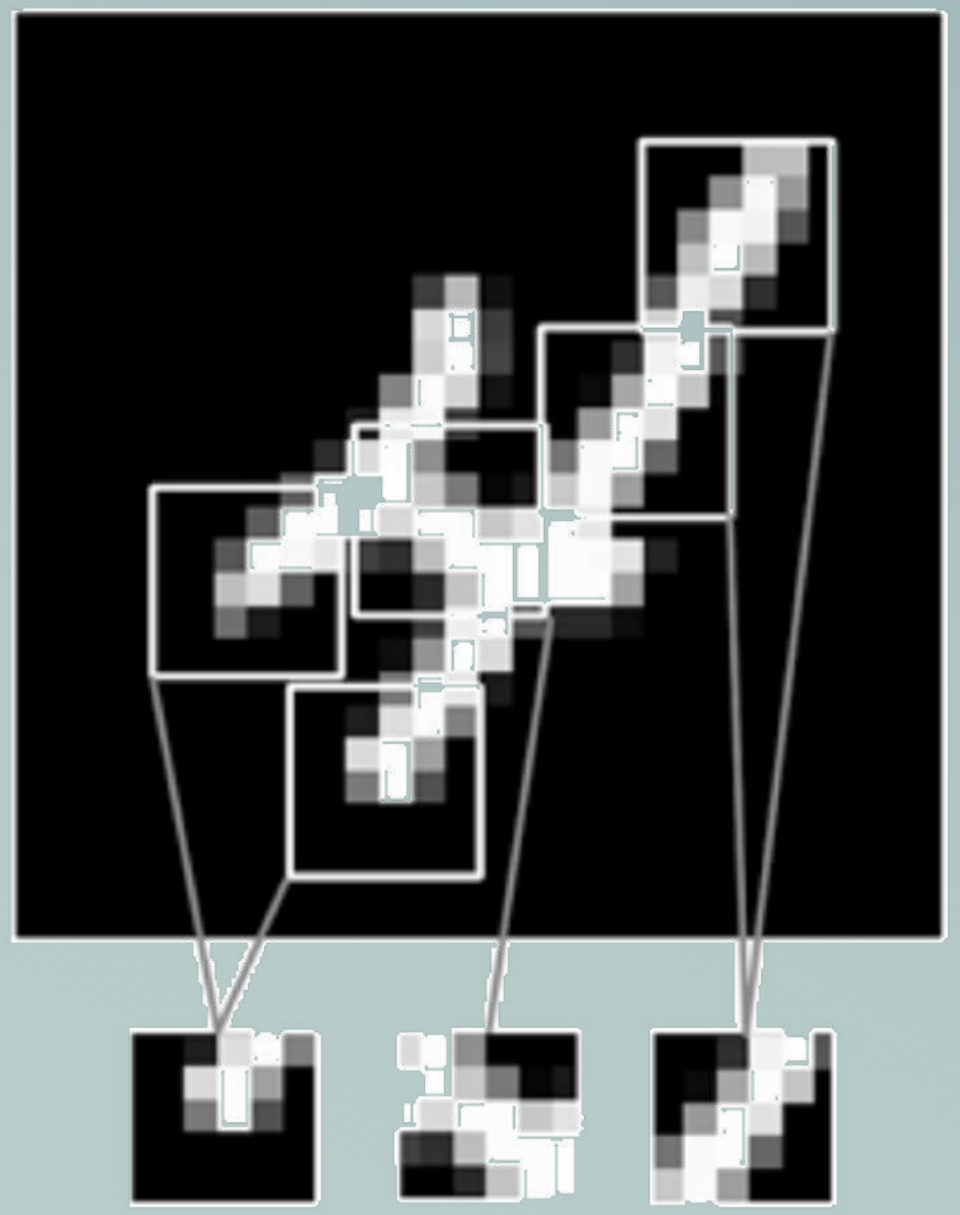




Links: https://blog.acolyer.org/2016/04/20/imagenet-classification-with-deep-convolutional-neural-networks/
Rechts: https://paperswithcode.com/sota/image-classification-on-imagenet?metric=Top%205%20Accuracy

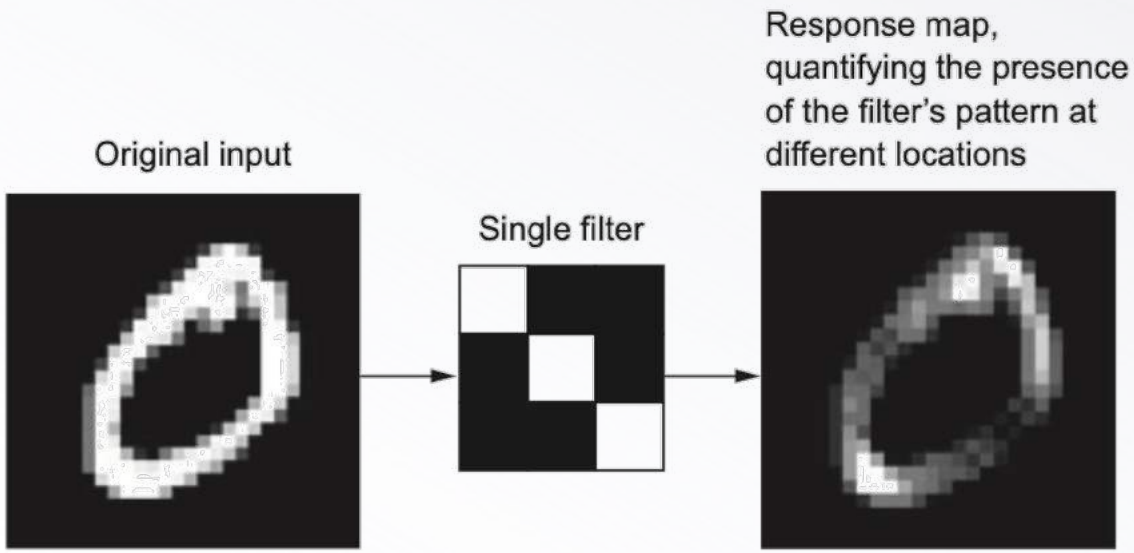
Why ConvNets?

- Dense layers can only learn global patterns, involving all pixels
- ConvNets have at two new key properties:
  - The learned patterns are **translation invariant**
  - They learn **spatial hierarchies of patterns**



ConvNets main building block: Filters

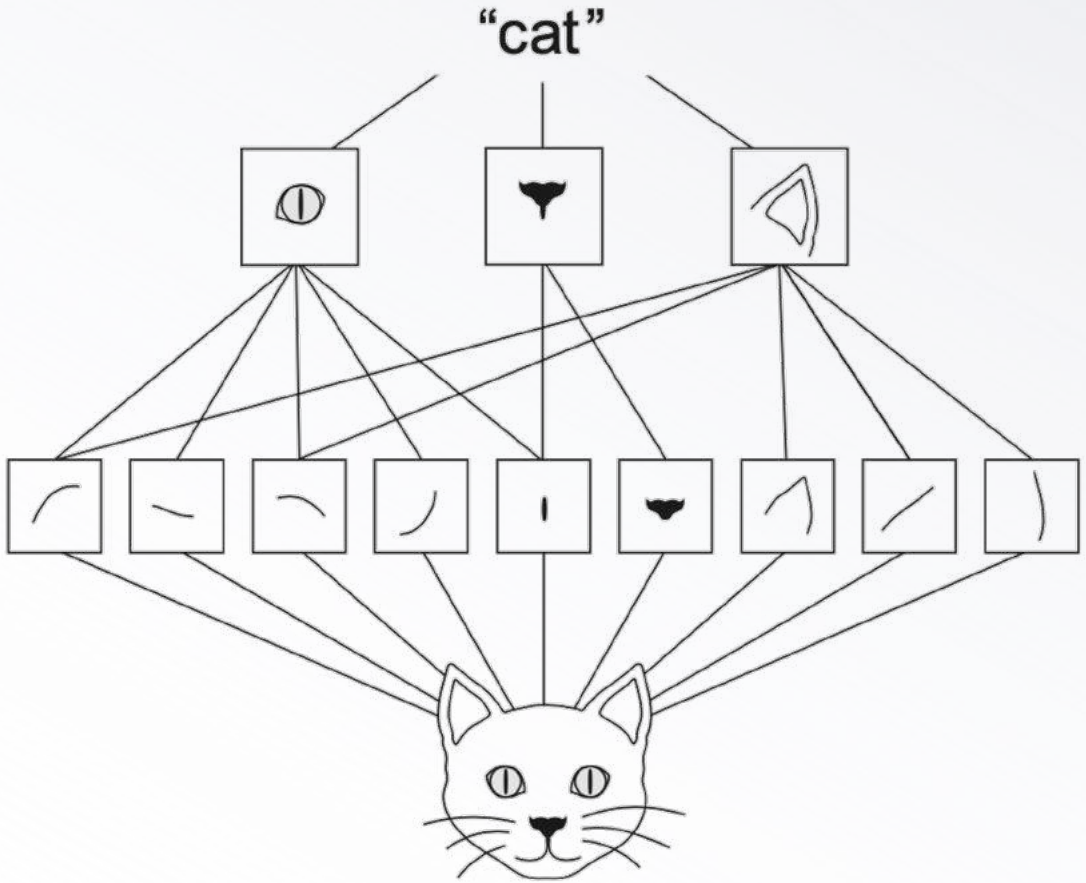
Encoding specific aspects of the input data



ConvNets learn increasingly complex and abstract visual concepts

larger patterns

small local patterns

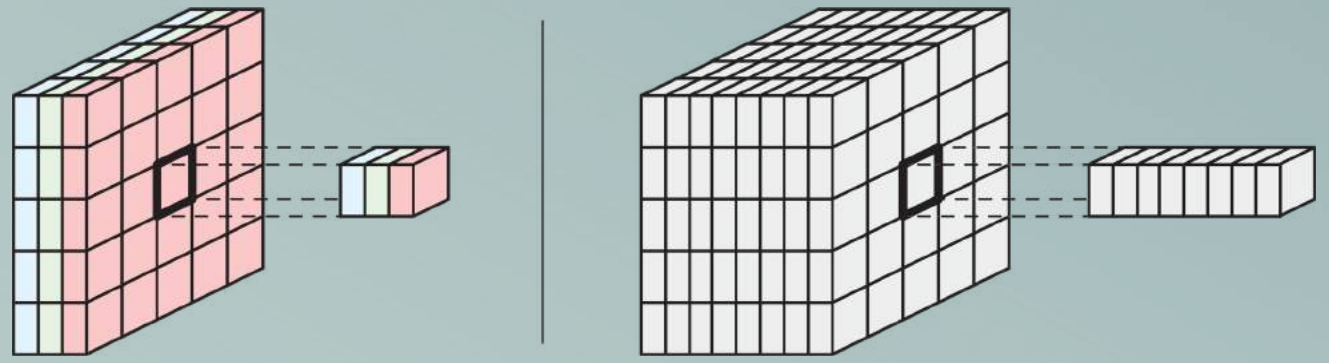


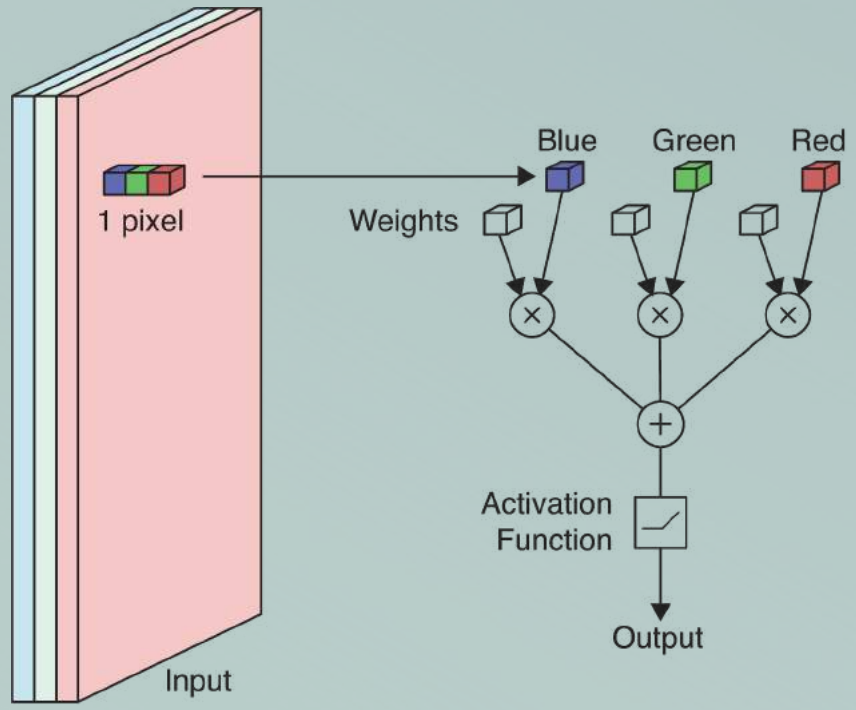
How convolution works

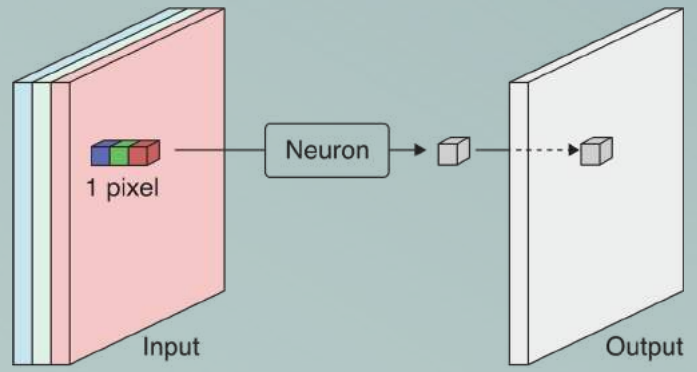
Image with one to three channels is made of up **pixels**

Tensor with any number of channels is made up of elements

We want to combine each input pixel's RGB values into a single number from 0 to 1 that represents "**yellowness**," which is the output pixel's value



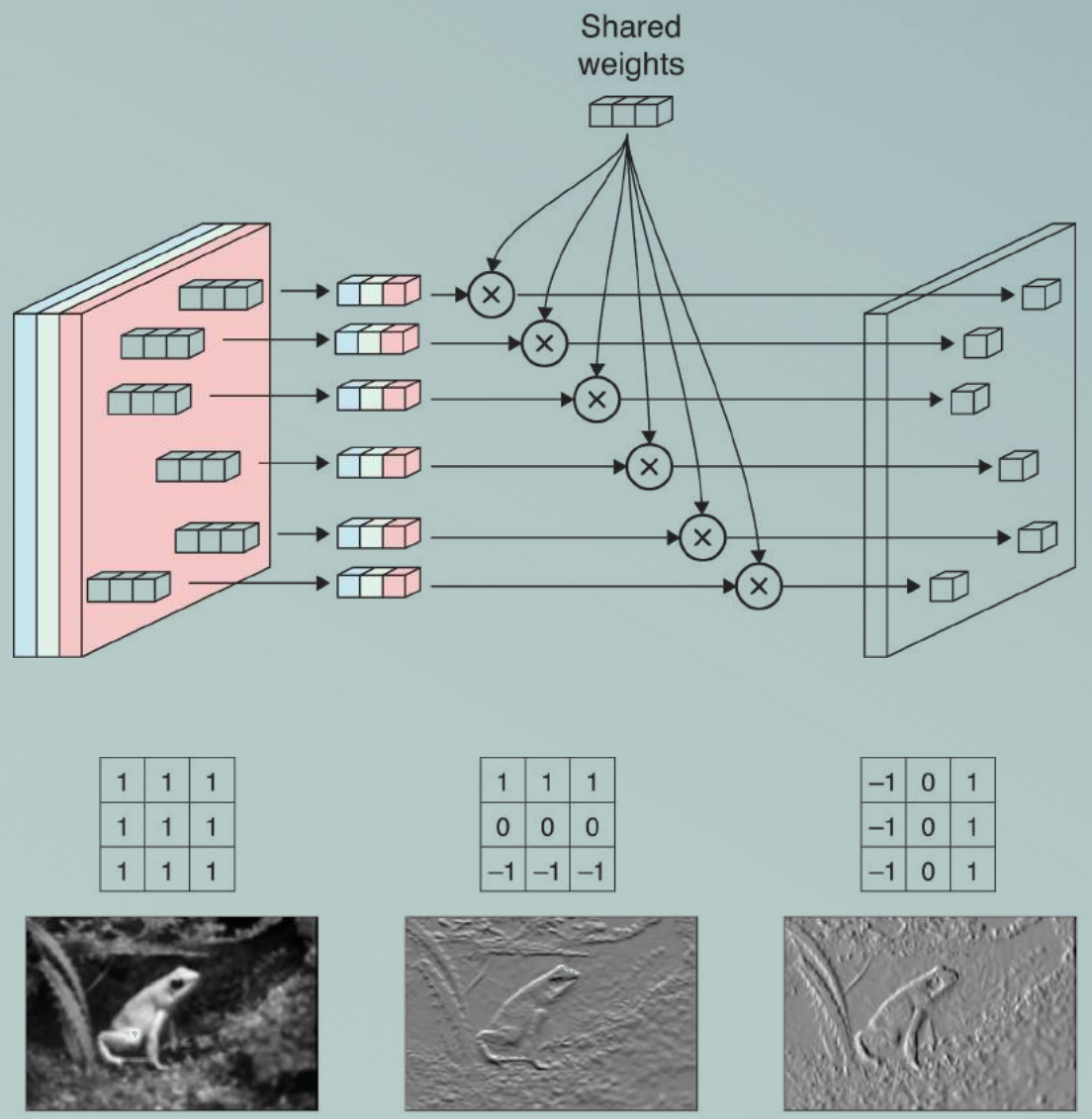




How convolution works

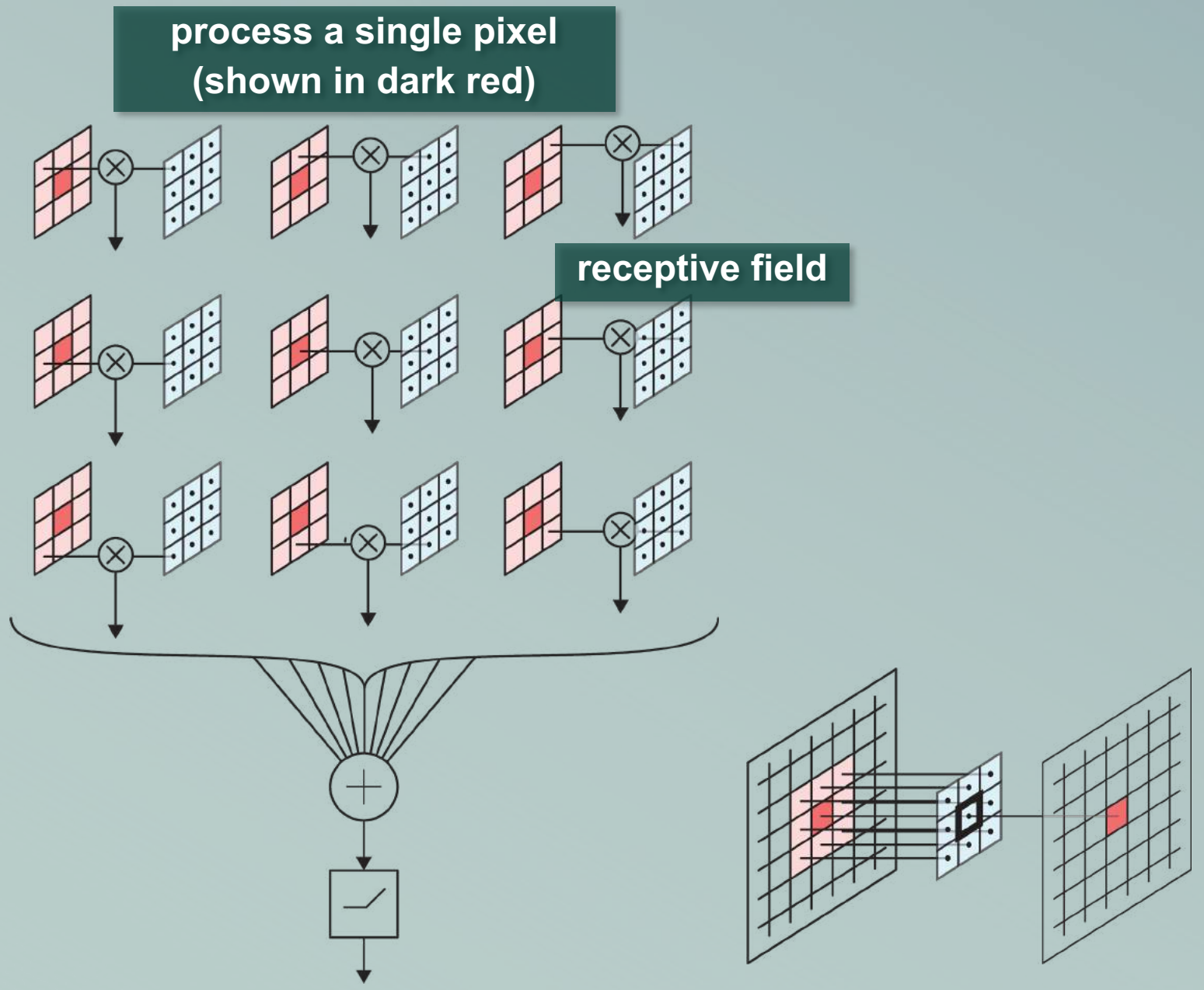
If we want to go faster, we can create a huge grid of identical neurons, and apply them to all the pixels **simultaneously**.

In many situations, it's also useful to look at the pixels near the one we're processing. Usually, we consider a pixel's eight immediate **neighbors**.



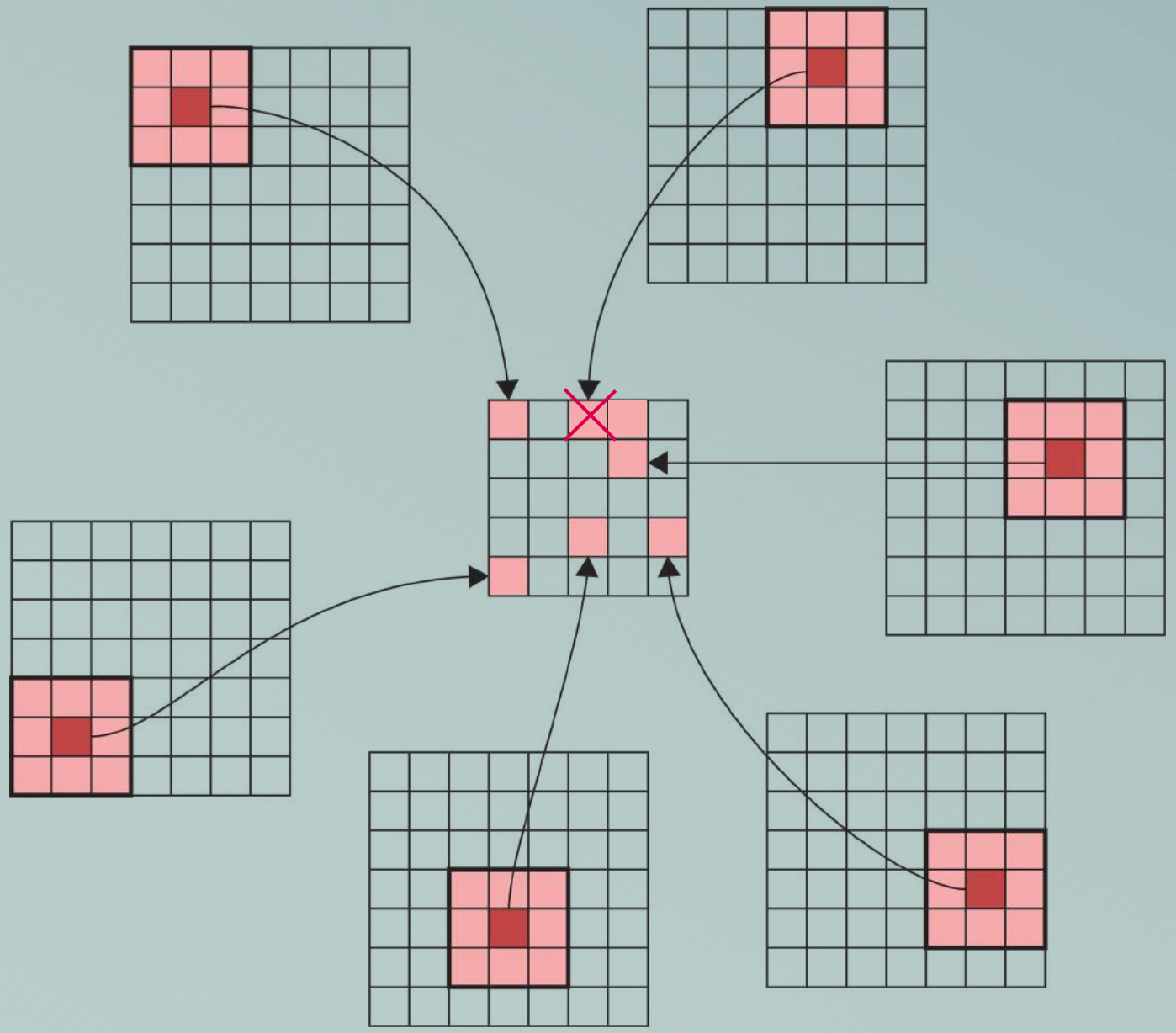
How convolution works

We can think of the blocks of numbers as weights, or **filter kernels**. In this scenario, we have a grid of nine weights that we place over a grid of nine pixel values. Each pixel value is multiplied by its corresponding weight, the results are summed up and run through an activation function, and we have our output



How convolution works

The figure shows the result of convolving a seven by seven input with a three by three filter. Note that if we were to center the filter over the input's corners or edges, the filter's footprint would extend beyond the input, and the neuron would require input values that aren't present.



How convolution works

Here we use a filter to find short, isolated vertical white stripes in a binary image.

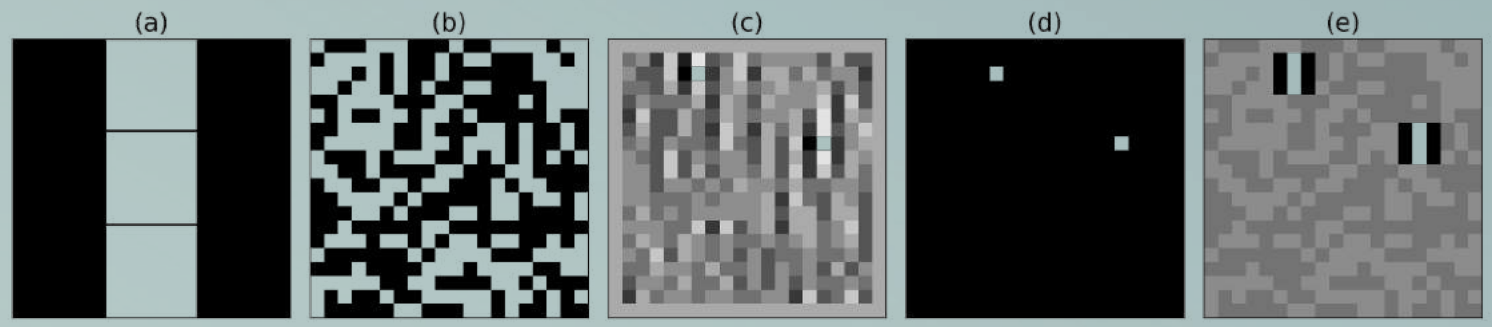
(a) shows a three by three filter

(b) shows a noisy input image

(c) shows the result of the convolution

(d) shows a thresholded version of (c)

(e) shows the area around the found pixels of (d) in (b)

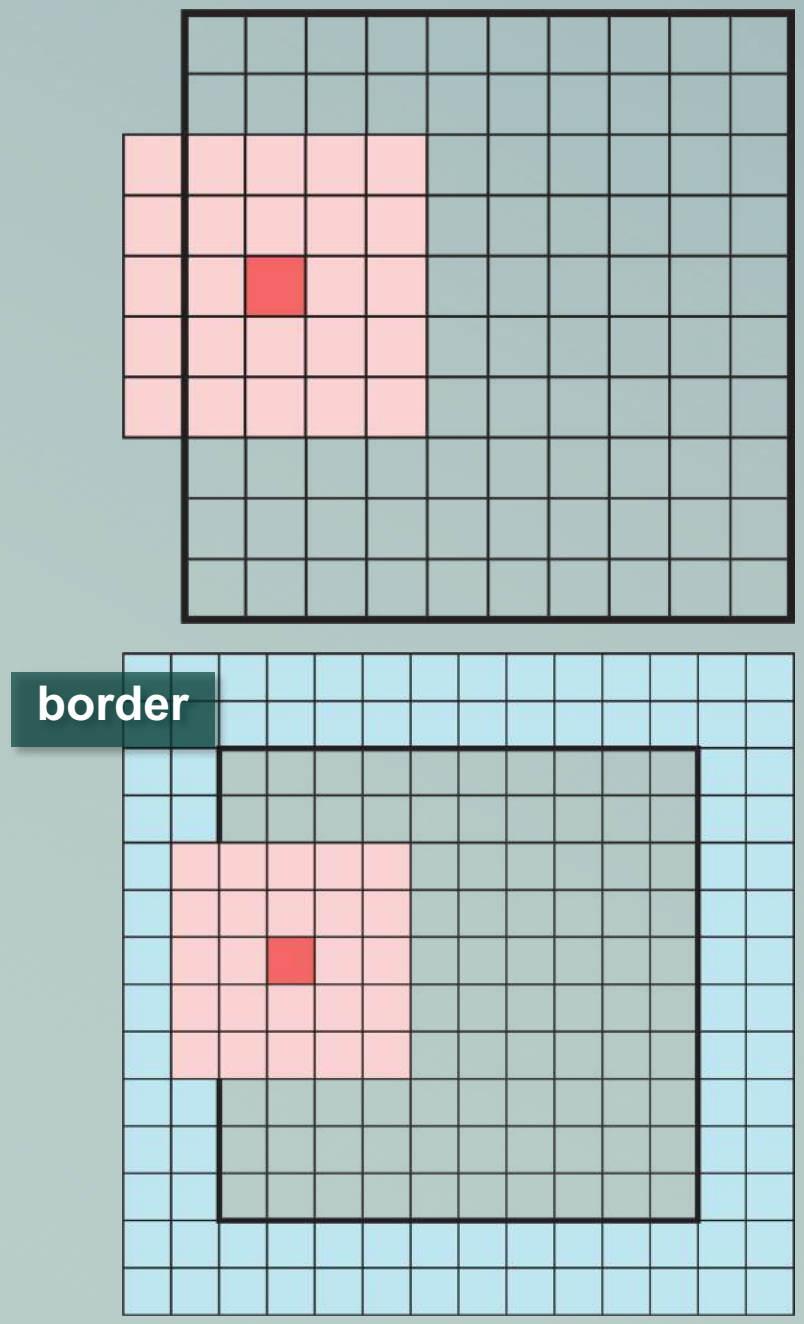


$$\begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ 0 & 0 & 0 \\ -1 & 0 & -1 \end{bmatrix} \Rightarrow -3 + 1 = -2$$

$$\begin{bmatrix} -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \Rightarrow 3$$

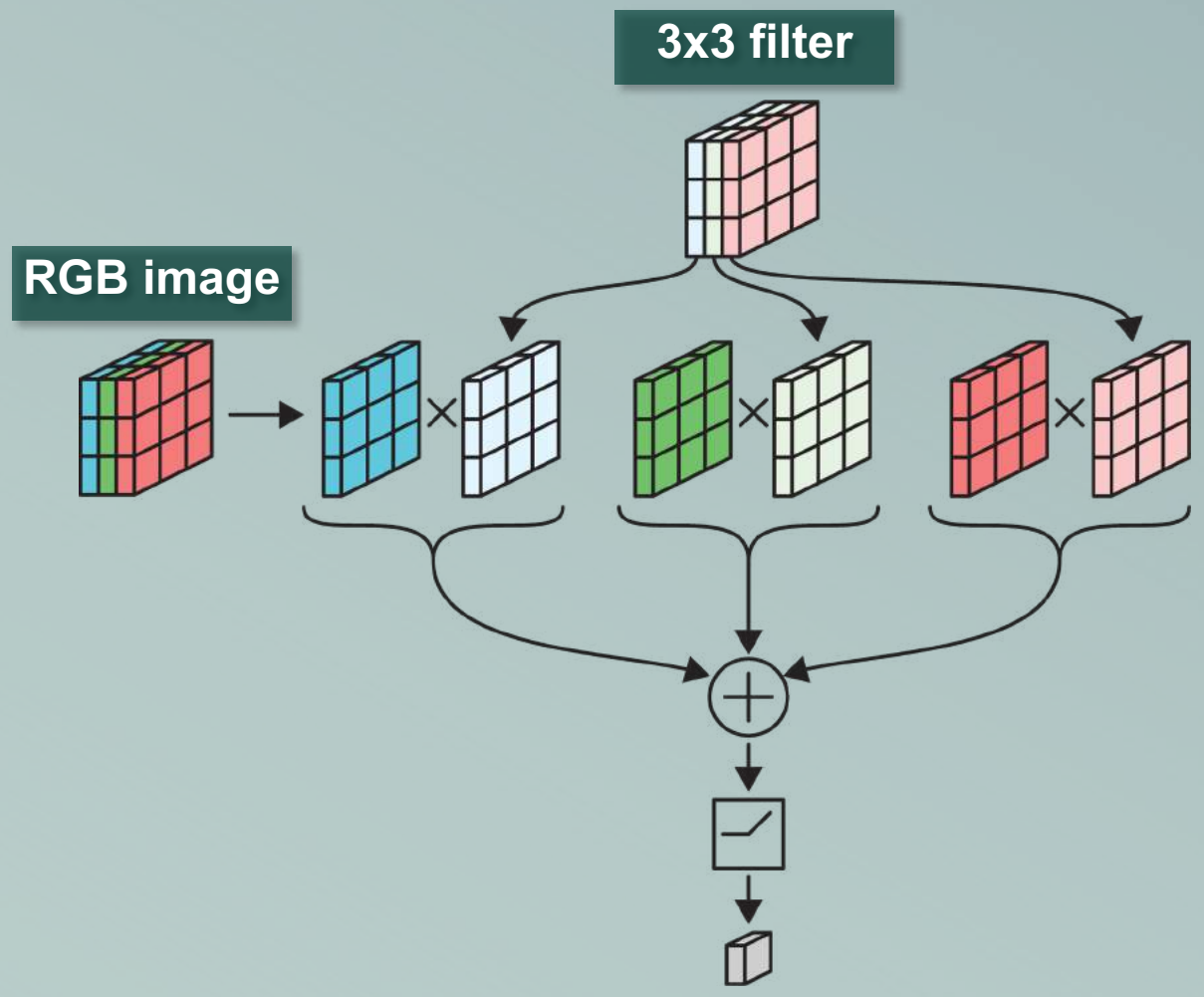
How padding works

Padding lets us create an output image of the same width and height as the input. The idea is that we add a border of extra elements around the outside of the input. All of these elements have the same value (zero-padding).



Multidimensional Convolution

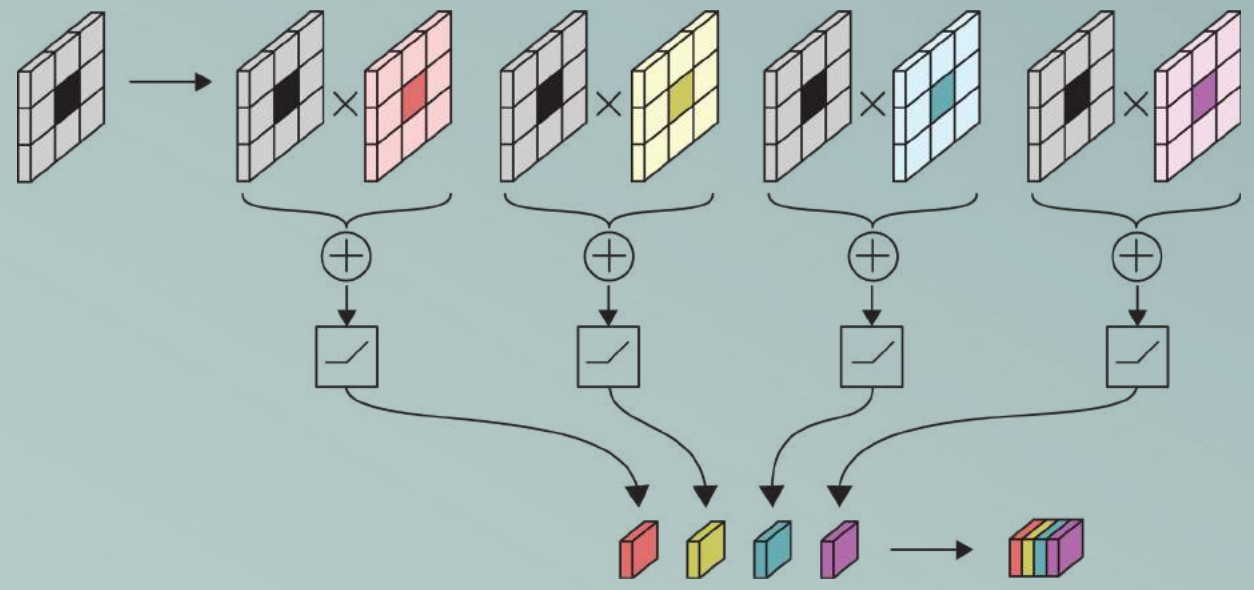
To process an input with multiple channels, our filters need to have an identical number of channels. That's because each value in the input needs to have a corresponding value in the filter. For an RGB image, a filter needs three channels. So, a filter with a footprint of three by three needs to have three channels, for a total of **27 numbers**.

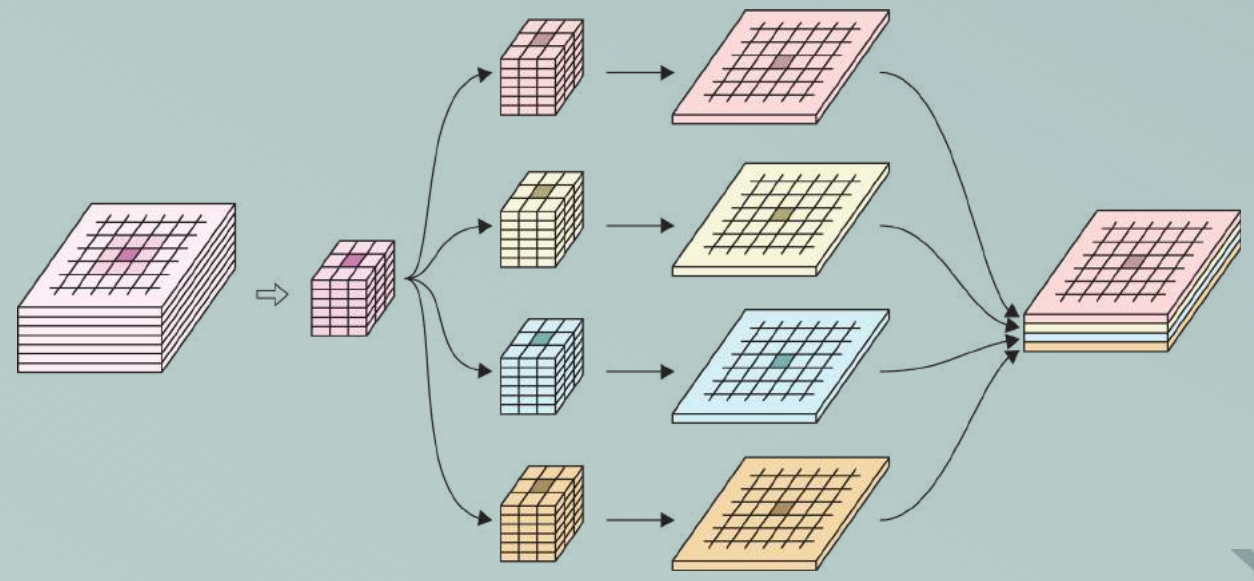


Multiple Filters

We've been applying a single filter at a time, but that's rare in practice. Usually, we **bundle up** tens or hundreds of **filters into one convolution layer** and apply them all simultaneously (and independently) to that layer's input.

The output of each filter is a **feature map** of a single channel. The output tensor is what we get from stacking these four feature maps, so it has four channels.

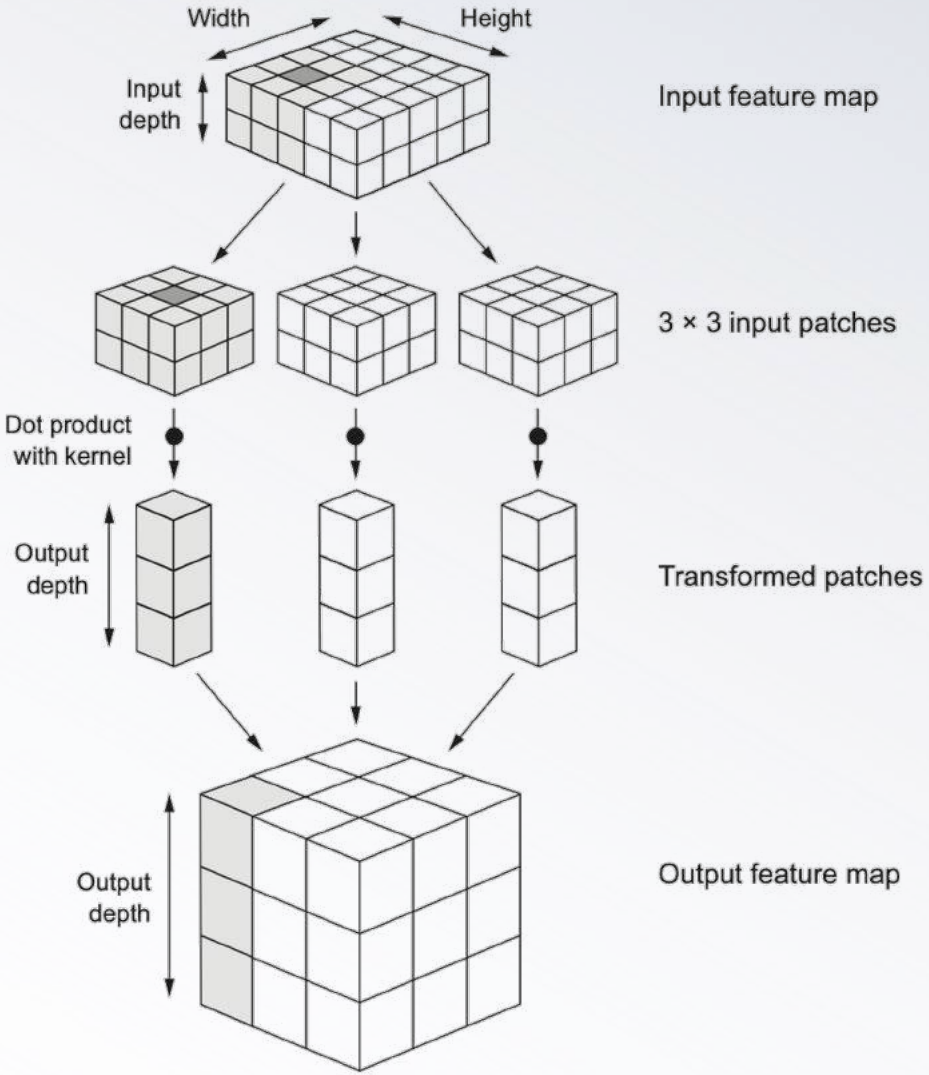




How convolution works

First, every filter in a convolution layer must have the same number of channels as that layer's input tensor.

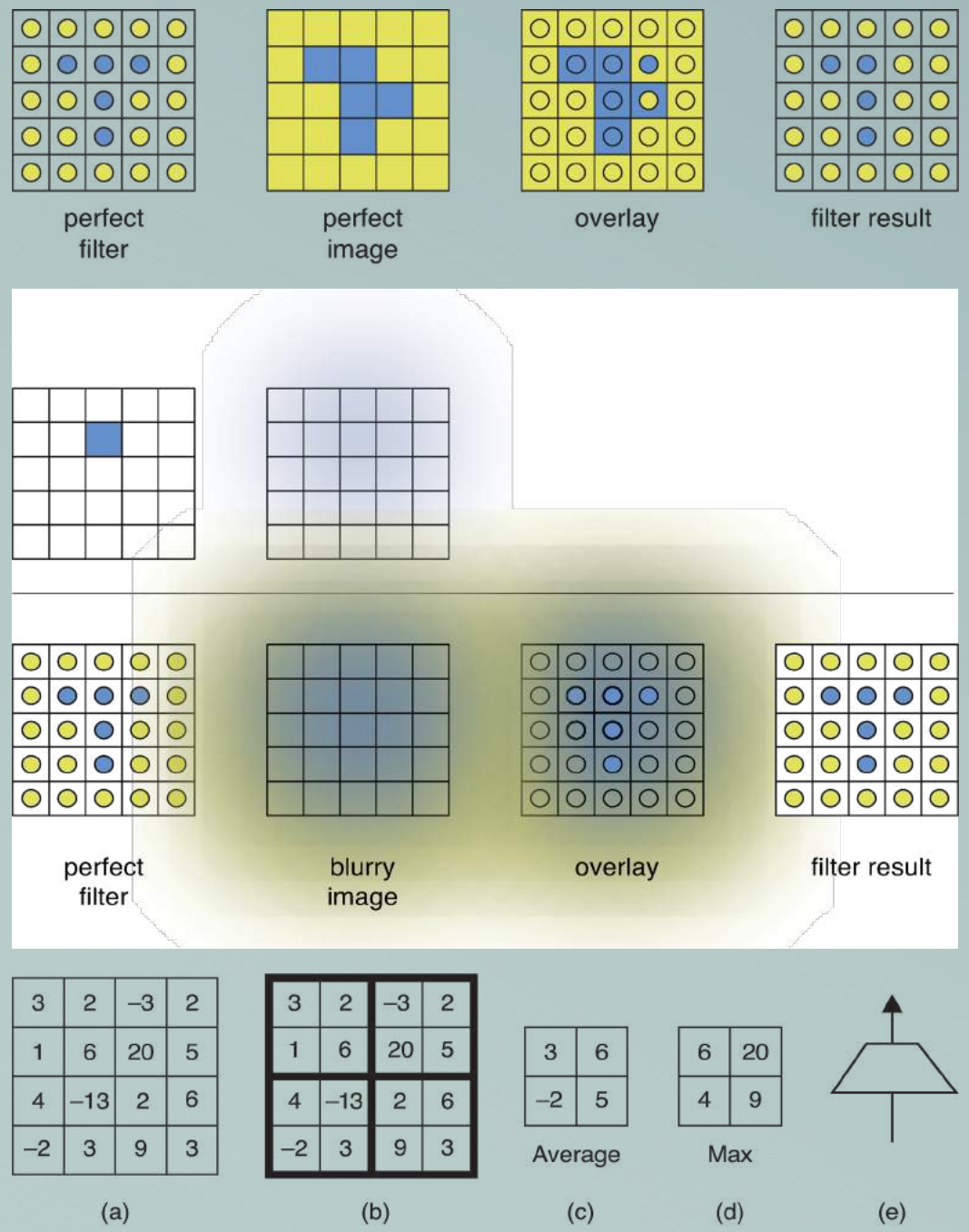
Second, a convolution layer's output tensor will have as many channels as there are filters in the layer.



Changing Output Size – Pooling

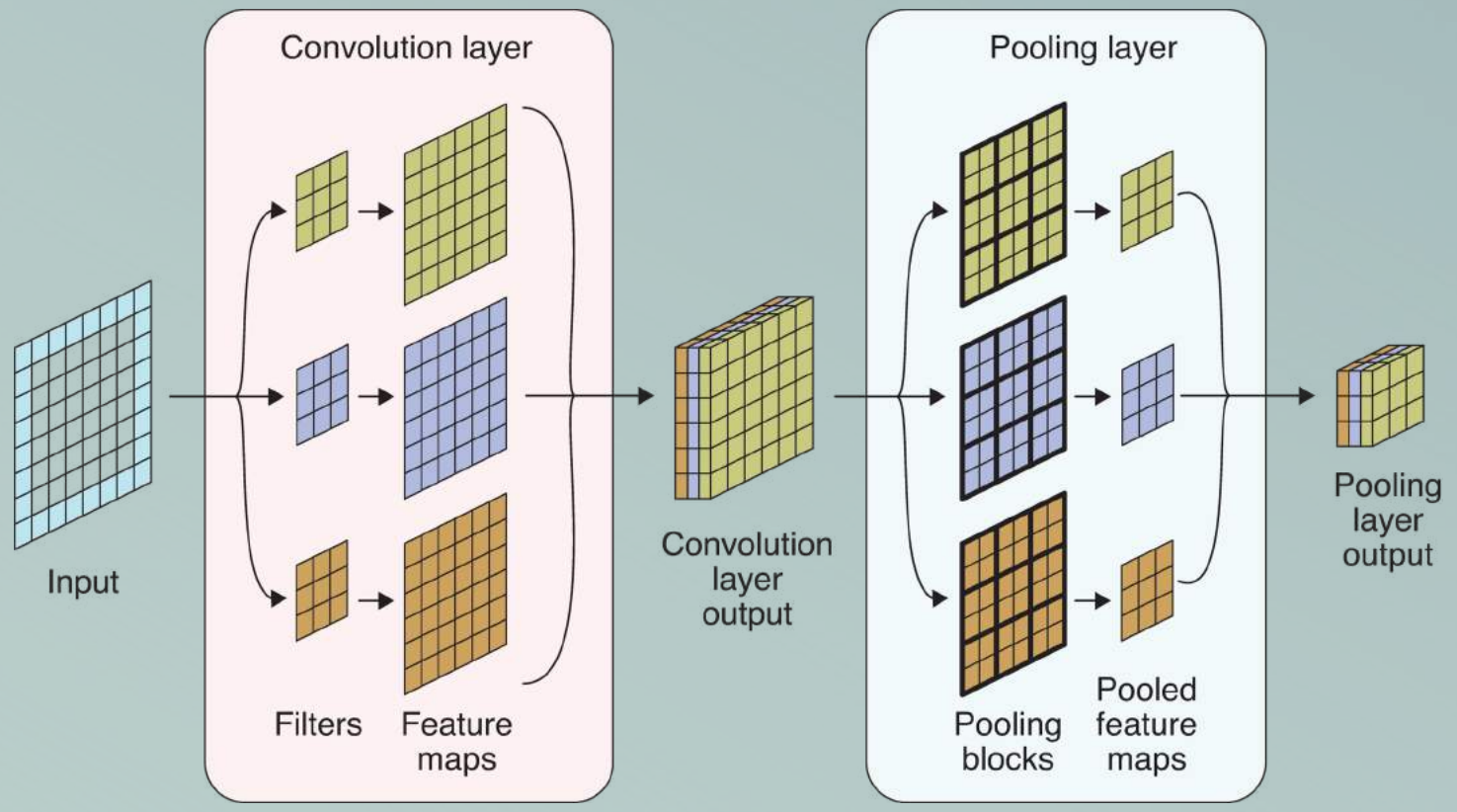
A filter matches the feature it's looking for if the underlying pixels match the filter's values.

We can come up with a technique to blur a tensor. We call the method **pooling**, or downsampling.



Changing Output Size

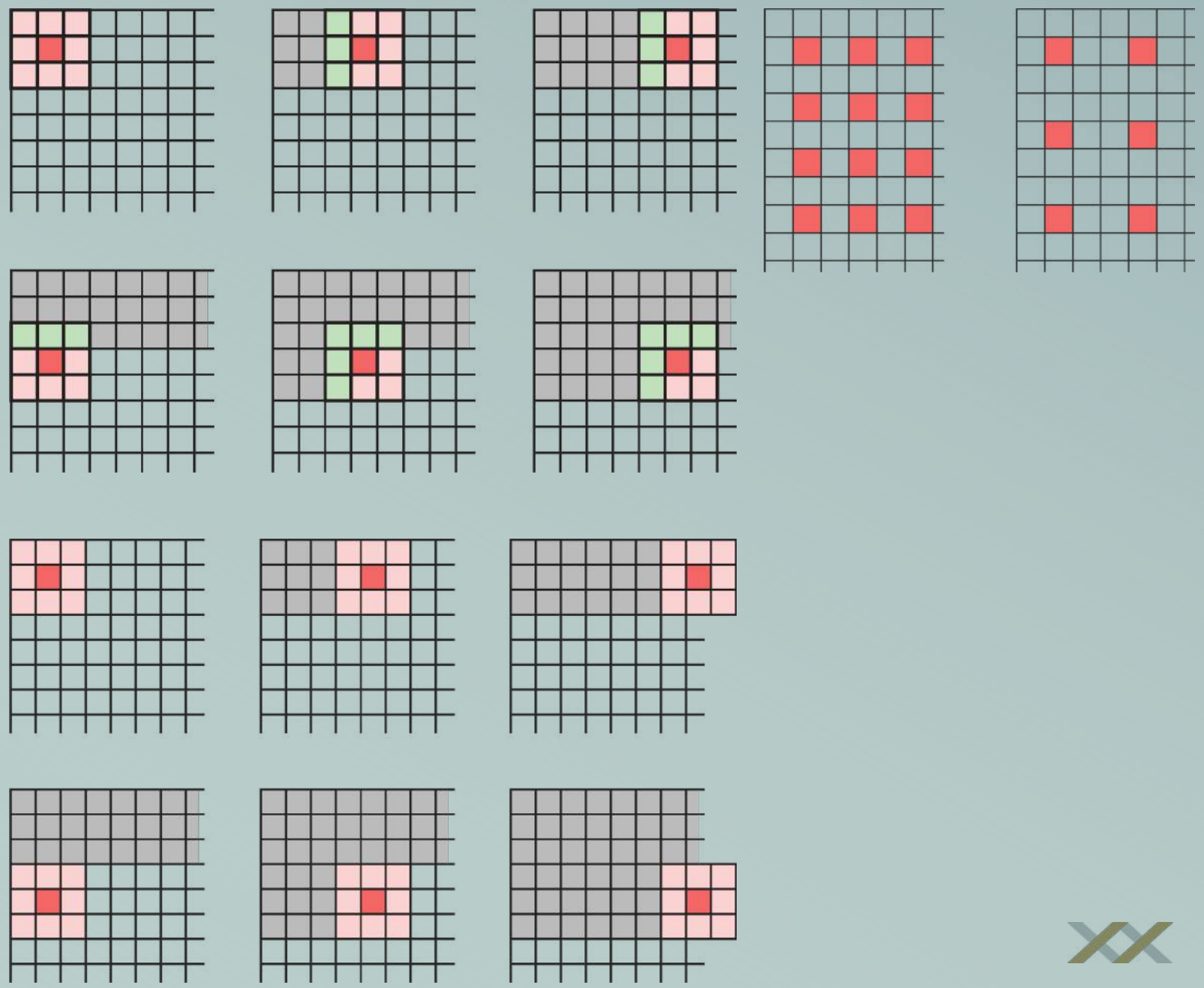
Pooling is a powerful operation that frees filters from requiring their inputs to be in precisely the right place. If some operation is insensitive to change of location it's called **invariant** with respect to that operation. Combining these, we say that pooling allows our convolutions to be **translationally invariant**, or **shift invariant.**



Changing Output Size – Striding

We've seen how useful pooling is in a convolutional network. Though pooling layers are common, we can save time by **bundling the pooling step right into the convolution process**. This combined operation is much faster than two distinct layers.

Strided convolutions are faster than convolution without striding followed by pooling.



Convolutional Nets in der Praxis

- Bilddaten einlesen und reskalieren
- Selber trainieren
- Features verwenden
- Fine tuning

PRAXIS

ConvNetJS – Deep Learning in your browser

7

Unsupervised Learning

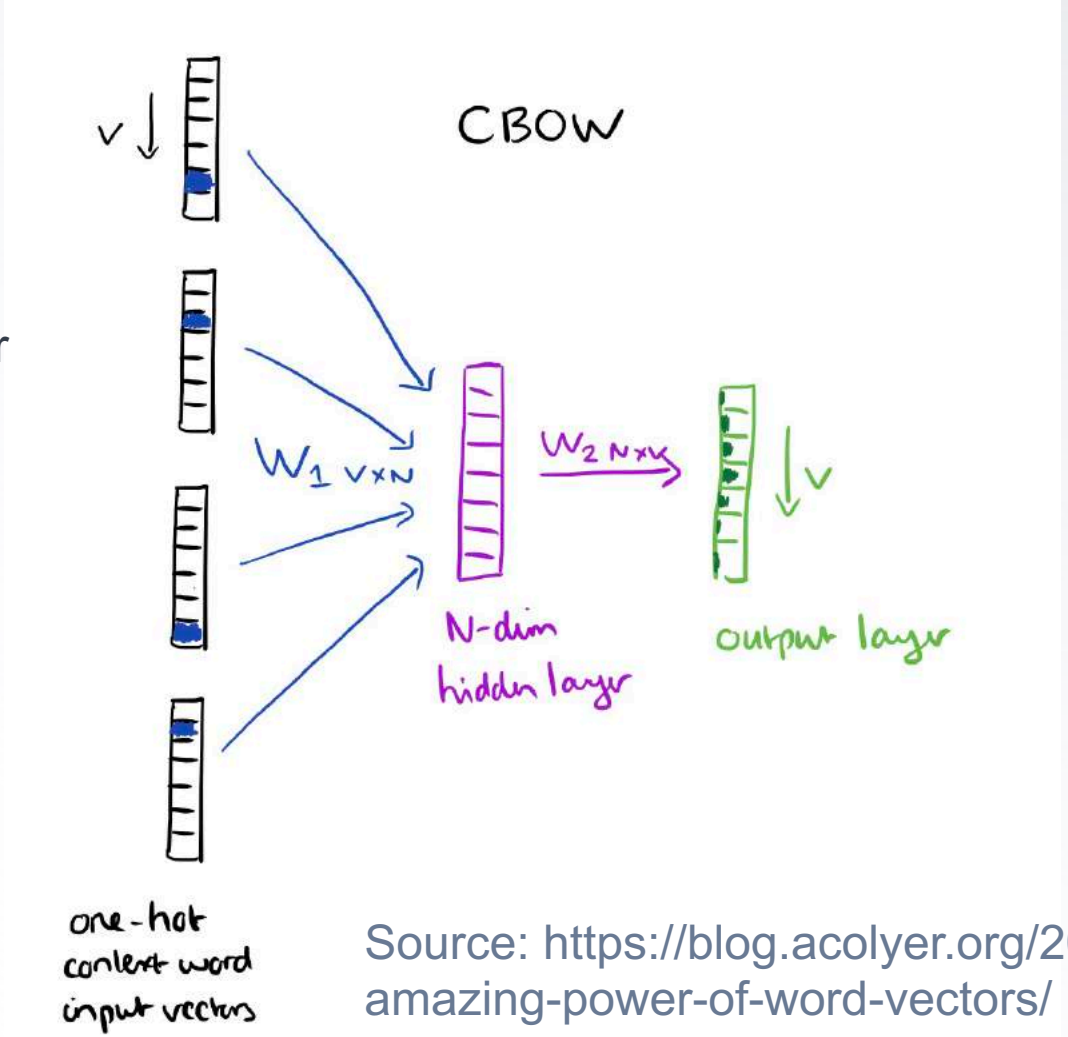
Autoencoder

Wortvektoren

Wörter kodieren jenseits von one-hot-encoding

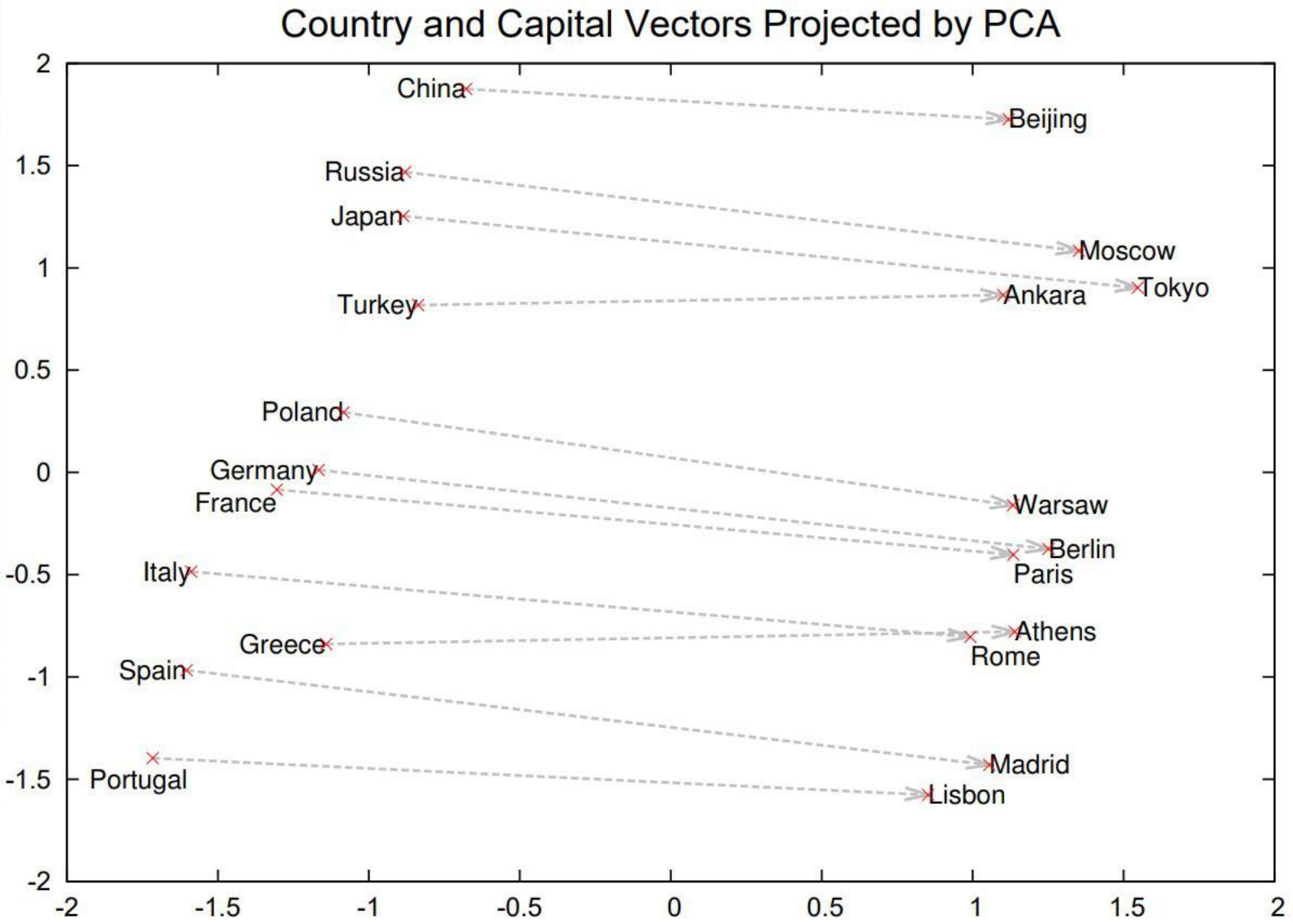
Die Idee: Wörter erhalten ihre Bedeutung aus dem Kontext ihres Gebrauchs. Also wenn man aus einer Vektorrepräsentation den Kontext vorhersagen kann …

One-hot-kodierte Kontextwörter werden in Wortvektoren umgewandelt und ein oder zwei weitere Layer sagen das Wort in ihrer Mitte vorher.



Source: https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

Wortvektoren kodieren semantische Informationen



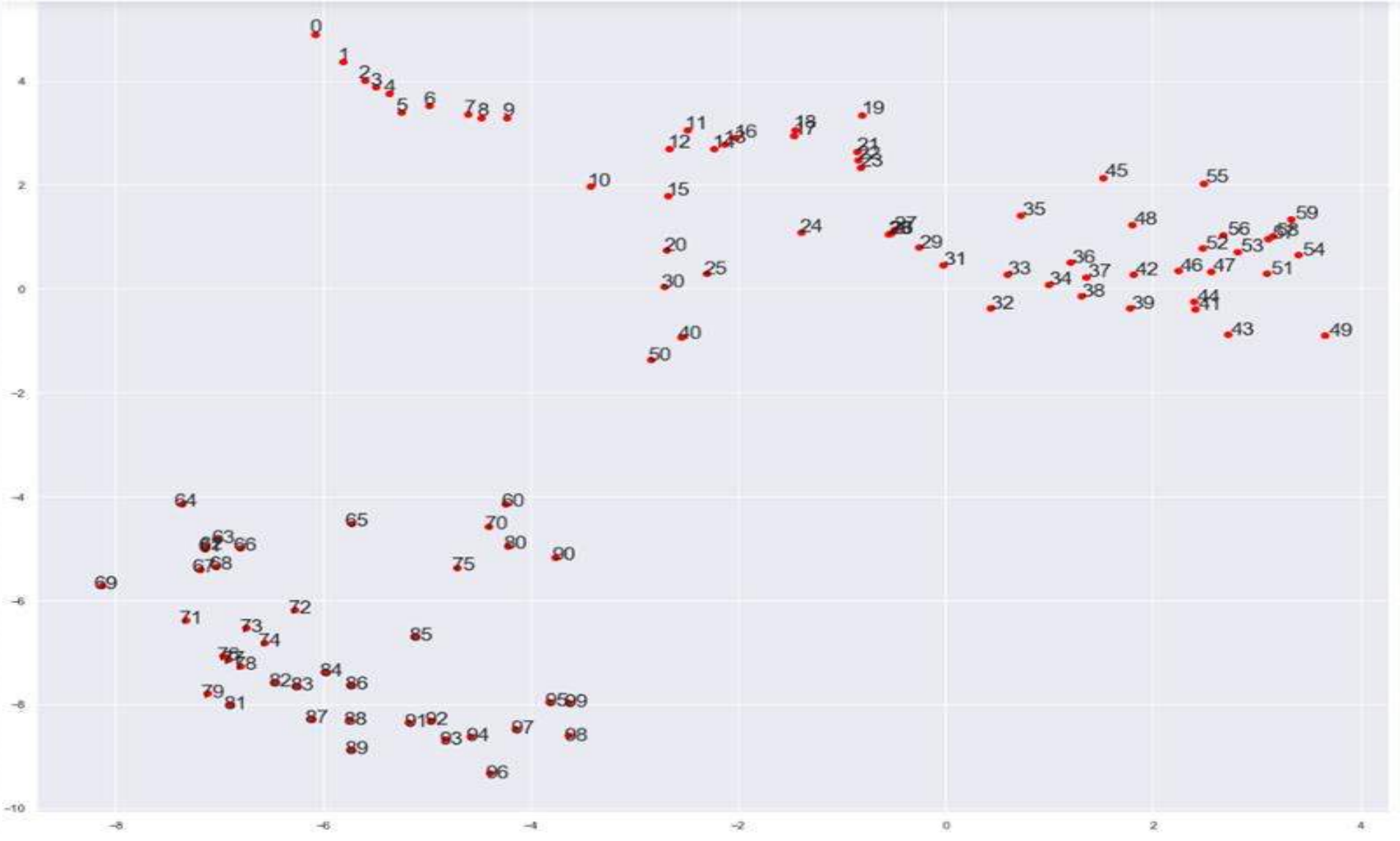
Source: Word2Vec-Paper

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

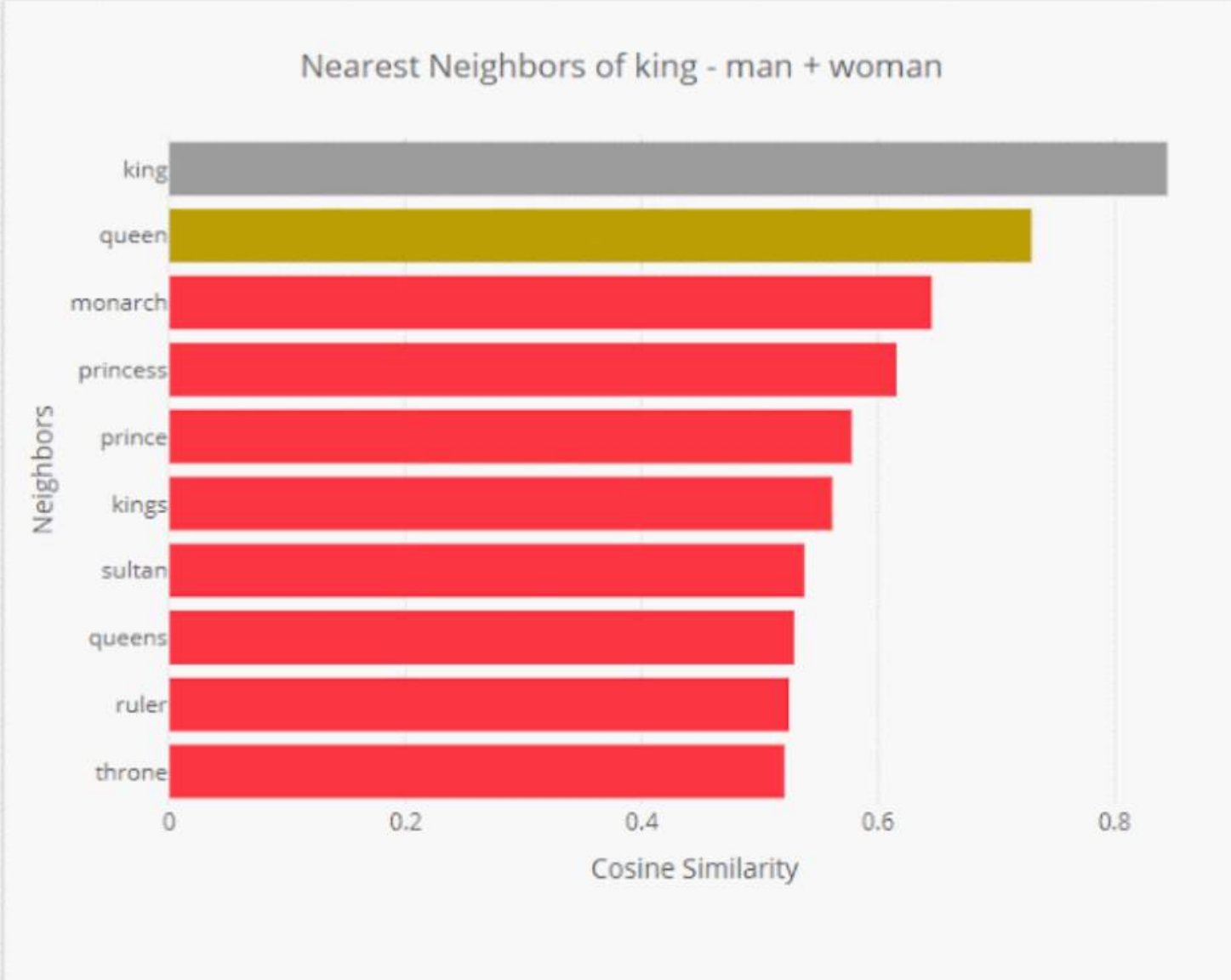
Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Source: Word2Vec-Paper

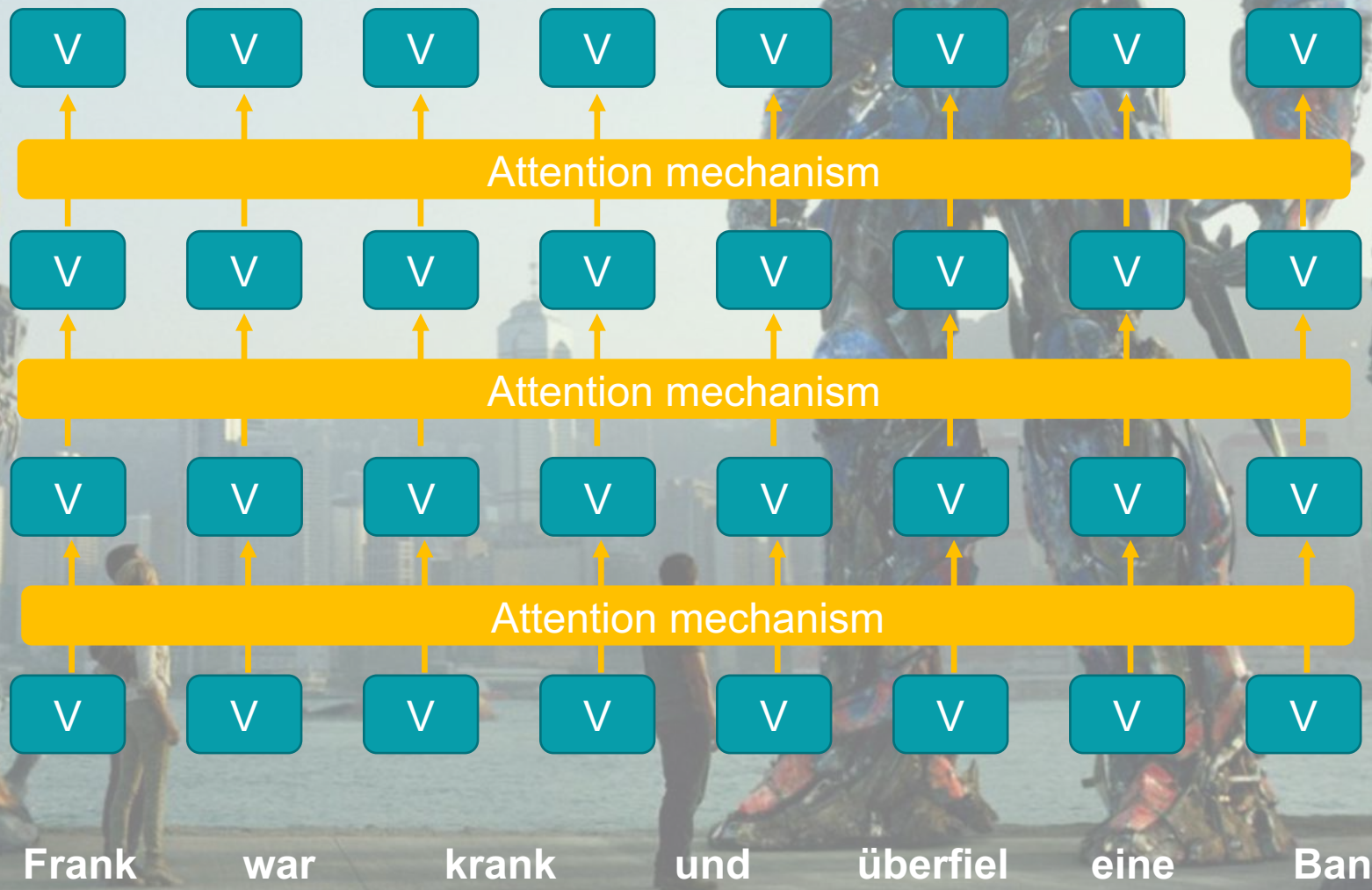
T-SNE der Wortvektoren von Zahlen



Wortvektoren kodieren relativ vage



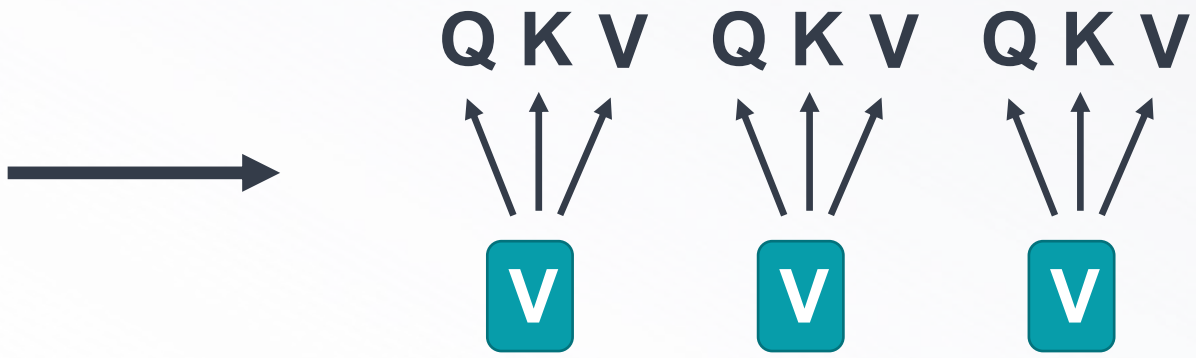
Transformer-Architektur



- Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017. (10000 citations!)

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the **query Q**, **keys K**, **values V**, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

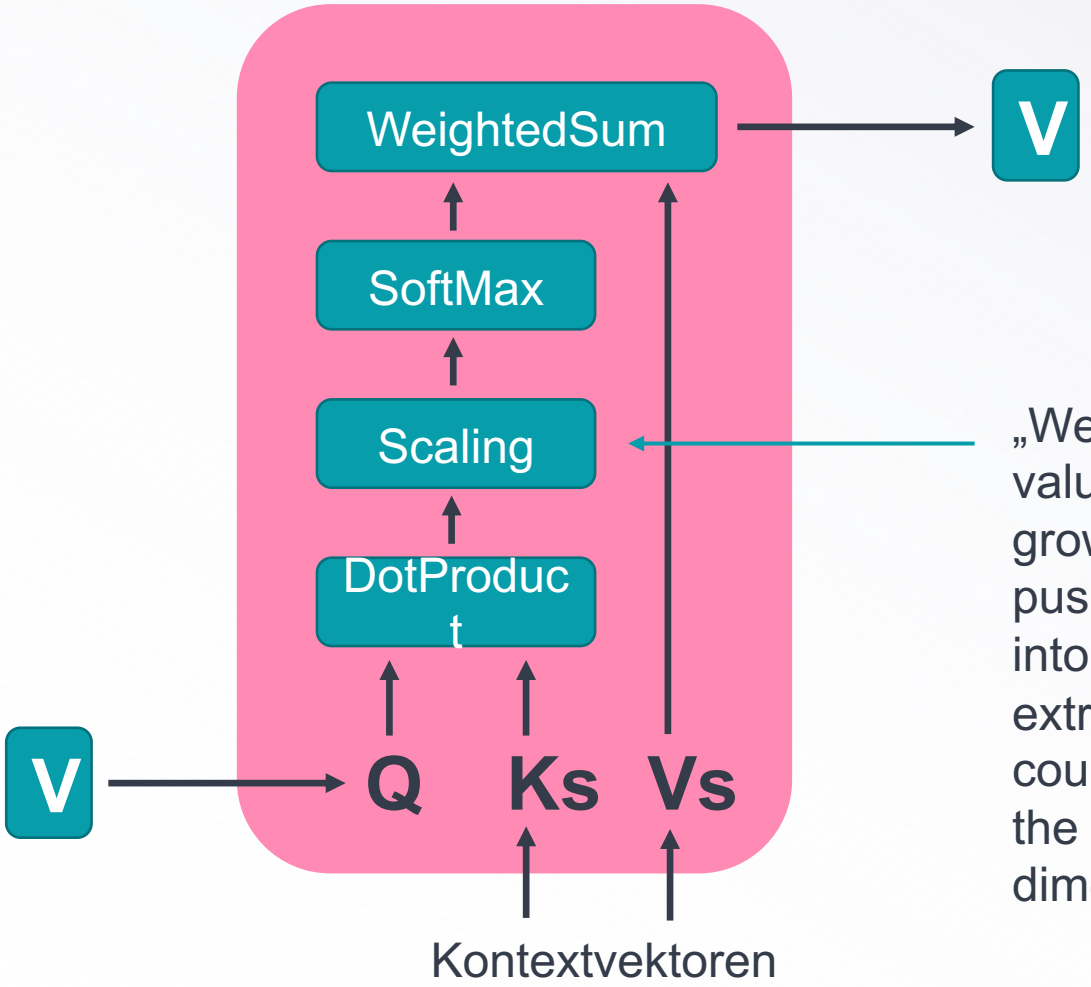
$$Q = VW^{Q}$$
$$K = VW^{K}$$
$$V = VW^{V}$$



Wie wird der Vektor verwurstet?

1. Er wird in einen Vektor **Q** verwandelt.
2. Dann mit den **K**eys des Kontextes verglichen.
3. Und dann aus den **V**alues des Kontextes gewichtet neu berechnet.

$$Q = VW^{Q}$$
$$K = VW^{K}$$
$$V = VW^{V}$$



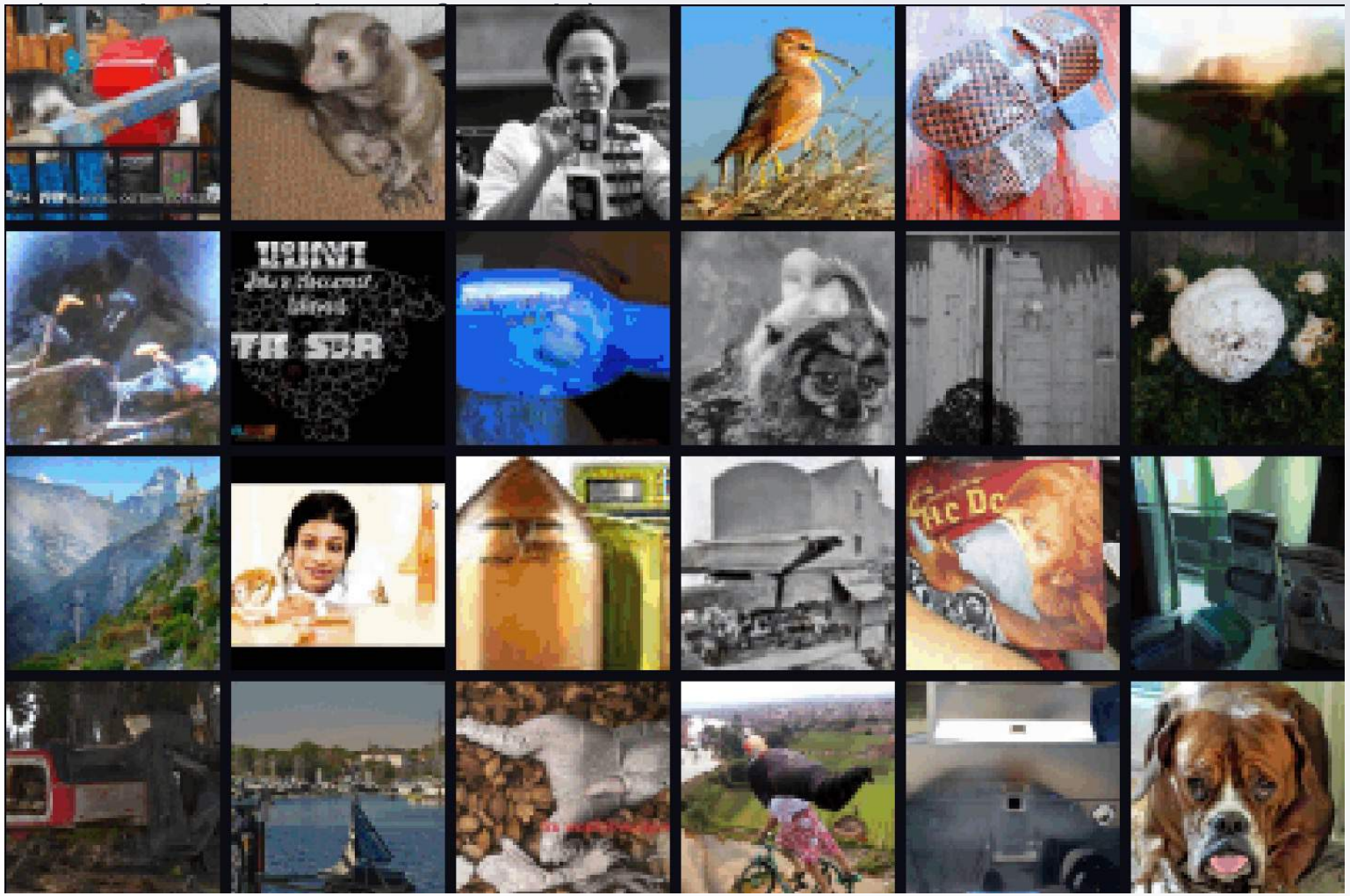
„We suspect that for large values of dk, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. To counteract this effect, we scale the dot products by 1/√dk. (dk = dimension of Q and K)“

Transformer Architektur – Charakteristiken

<u>Anwendungsgebiete:</u>
- Neural Machine Translation
- Sprachmodelle
- Sequence learning/generation

<u>Training:</u>
- Autoregressive für Generation
- Masked für NLP-Tasks
- Finetuning für spezifische Tasks

<u>Versionen:</u>
- Sparse Transformer (OpenAI)
- Compressive Transformer (Deepmind)
- Reformer (GoogleAI)
- Linformer (Facebook)



EXXETA
CONSULTING AND TECHNOLOGIES

DATA SCIENCE PPT TEMPLATE

Aus großer Kraft,
folgt große Verantwortung

AUF EINEN BLICK

DAS IST EXXETA

Unabhängiges, mittelständisches Beratungs- und Lösungsunternehmen mit den Schwerpunkten IT-Entwicklung & Beratung, Methoden- und Innovationsmanagement.

- gegründet 2005
- > 850 Mitarbeiter
- 90,5 Millionen € Umsatz
- 12 Standorte (GER, CH, SK)

Für den Erfolg jedes einzelnen Kunden

verbindet EXXETA

BUSINESS & TECHNOLOGY

auf innovative Weise

FÜR UNSERE KUNDEN

DIGITALE PRODUKTE UND SERVICES

EXXETA bietet maßgeschneiderte Lösungen für verschiedene Plattformen aus den Bereichen Strategie und IT für die Digitalisierung großer und mittlerer Unternehmen.

STRATEGISCHE ORIENTIERUNG

Gemeinsame Standortbestimmung mit Zielsetzung, Definieren des strategischen Rahmens und Identifizieren von Hürden und Herausforderungen.

KONZEPT & DESIGN

Schnelles und iteratives Generieren, Konzeptionieren und Testen von Lösungsideen basierend auf fundiertem Nutzerfeedback.

PROTOTYPEN & IMPLEMENTIERUNG

Iteratives Prototypisieren des Lösungskonzeptes mit ständigem Nutzerfeedback und technische Implementierung mit Scrum.

BETRIEB & WARTUNG

Sicherstellen von Stabilität und Performance der Anwendung, qualifizierter Support, Fehleranalysen und Durchführen von Updates.

WE ❤ DATA

data.exxeta.com